Untersuchungen zu Datentyperweiterungen für XML-Dokumente und ihre Anfragemethoden am Beispiel von DB2 und Informix

Diplomarbeit

Universität Rostock, Fachbereich Informatik

vorgelegt von: Beate Porst
geboren am: 17. April 1970 in Neuruppin

Betreuer: Prof. Dr. Andreas Heuer
Dr.-Ing. Holger Meyer
Dr.-Ing. Meike Klettke
Dipl.-Inf. Joachim Kröger

Bearbeitungszeitraum: 01.11.1998 - 30.04.1999

Zusammenfassung

Die Integration der Typen *Binary Large Object* (BLOB) und *Character Large Object* (CLOB) ermöglicht die Verwaltung großer Objekte in einer Datenbank. Insbesondere Anwendungen wie "Digitale Bibliotheken" nutzen diese Datentypen. Durch die Verwaltung großer Objekte in der Datenbank kommt der Retrieval-Funktionalität hier eine wachsende Bedeutung zu. Neben den unstrukturierten Textobjekten existiert eine Dokumentenkategorie, die eine implizite Struktur aufweist, die aber aufgrund der Heterogenität nicht durch ein Datenbankschema beschrieben werden kann. Das Problem der existierenden Retrieval-Module ist nun, daß der Schwerpunkt ihrer Anfragen auf der reinen Inhaltssuche beruht. Anfragen, die hingegen die Struktur sowie eine Kombination aus Struktur und Inhalt betreffen, werden nicht berücksichtigt.

In dieser Diplomarbeit wird untersucht, wie die Volltext-Retrieval-Erweiterungen der Datenbanksysteme DB2 und Informix genutzt werden können, um XML-Dokumente zu verwalten und mit Anfragemethoden finden zu können. Dazu wurde zunächst die Funktionalität des TextExtenders und des Excalibur Datablade evaluiert, um anhand von Anforderungen die Eignung für das Retrieval semistrukturierter Dokumente abzuleiten. Aufbauend auf dem Suchausdruck des DB2 TextExtenders wurde eine Konzeption erarbeitet, die es ermöglicht, ein Retrieval auf semistrukturierten Dokumenten vorzunehmen, wobei eine Differenzierung zwischen Struktur und Inhalt erfolgt. Der abschließende Vorschlag eines allgemeinen XML-Datentyps versucht Einschränkungen und Probleme dieser Konzeption und existierender XML-Anfragesprachen zu vermeiden.

Abstract

The integration of BLOB and CLOB types allows the management of large objects in a database. Particularly applications such as "Digital Libraries" use these data types. Due to the management of large objects in a database, retrieval functionality becomes increasingly important. Besides unstructured text objects, there exists a class of documents that have an implicit structure. But as a consequence of heterogeneity, this structure cannot be described by a database schema. The drawback of traditional retrieval modules is the fact that the priority of their queries is based on pure content search. They do not support queries on structure or a combination of structure and content.

In this master thesis we examined how to use full text retrieval extensions of DB2 and Informix for managing and finding XML documents. First, the functionality of TextExtender and Excalibur DataBlade has been evaluated. The purpose was to determine the applicability of these full text modules for retrieval on semistructured data. Based on the DB2 TextExtender search syntax, we designed a solution, which distinguishes between structure and content. Thus,

Information Retrieval queries on semistructured data can be executed. Finally, due to restrictions and problems of this concept and other existing XML query languages, a universal XML data type was proposed.

CR-Klassifikation

H.2.	Database Management
H.2.1.	Data Models
H.2.3.	Query Languages
H.3.3.	Information Search and Retrieval
I.7.2.	Standard Text Processing

Key Words

XML, objektrelationale Datenbanken, semistrukturierte Daten, Information Retrieval

Inhaltsverzeichnis

Kapitel 1

Einleitung

Mit der weltweiten Zunahme an elektronisch publizierten Dokumenten wird Datenbanken zur Speicherung, Verwaltung und Manipulation dieser Dokumente eine größer werdende Aufmerksamkeit gewidmet. Für zahlreiche Anwendungen, wie Digitale Bibliotheken, Online-Nachschlagewerke etc., ist insbesondere die Informationsgewinnung aus diesen Dokumenten ein wichtiges Kriterium. Volltext-Retrieval-Systeme stellen hier eine populäre Möglichkeit zur Informationsextraktion dar. Neben reinen Textdateien existiert eine Dokumentenkategorie, die eine implizite Struktur aufweist, die aber aufgrund ihrer Heterogenität nicht durch ein Datenbankschema beschrieben werden kann. Das Problem traditioneller Volltext-Retrieval-Systeme ist nun, daß der Schwerpunkt ihrer Anfragen auf der reinen Inhaltssuche beruht. Anfragen, die hingegen die Struktur sowie eine Kombination aus Struktur und Inhalt betreffen, werden nicht unterstützt, obwohl hier ein großes Potential zur Verbesserung der Retrieval-Qualität liegt.

Die beiden Projekte GETESS und Global-Info (siehe Abschnitt 1.1 und 1.2) befassen sich unter anderem mit dem oben genannten Problem des Text-Retrievals in semistrukturierten Dokumenten. Da die in dieser Diplomarbeit konzipierte Datentyperweiterung auch ein Ansatz für diese Projekte sein könnte, sollen die Ziele und Aufgaben der Projekte (im speziellen der Teilprojekte der Universität Rostock) hier kurz vorgestellt werden.

1.1 GETESS

Gegenstand des BMBF[1]-Forschungsprojektes GETESS (*GE*rman *T*ext *E*xploitation and *S*earch *S*ystem) ist die Entwicklung eines intelligenten Werkzeugs zur Informationsrecherche im Internet.

[1]Bundesministerium für Bildung, Wissenschaft, Forschung und Technologie

Aufgrund der Mächtigkeit der Informationsmenge im World Wide Web (WWW) wird es immer schwieriger, gezielt Informationen zu finden. Suchmaschinen wie AltaVista arbeiten bei ihrer Suche größtenteils mit syntaktischen Methoden, deren Beschränkungen jedoch zu unbefriedigenden Ergebnissen führen. Das Projekt GETESS widmet sich nun dieser Problematik, indem es zur Verbesserung der Ergebnisse des Information-Retrieval die Semantik von WWW-Dokumenten mit Hilfe eines eingeschränkten natürlich-sprachlichen Prozessor extrahiert und aus diesen Informationen *abstracts* bildet. Die GETESS-Suchmaschine wird dann in der Lage sein, neben den herkömmlichen Informationen auch die in einer Datenbank gespeicherten *abstracts* mit in die Suche einzubeziehen und, falls erforderlich, zwischen den *abstracts* Beziehungen herzustellen.

Das Gesamtsystem GETESS besteht aus folgenden Kernteilen:

Dialogsystem: Das Dialogsystem soll sowohl die natürlich-sprachliche Interaktion mit dem Nutzer, als auch formale Anfragen, die in der Regel von erfahrenen Anwendern bevorzugt werden, unterstützen.

Suchmaschine: Kernpunkt des GETESS-Systems bildet eine Suchmaschine, die neben herkömmlichen Informationen, in der Lage sein muß, Informationen aus Datenbanken mit in die Suche einzubeziehen. Diese Suchmaschine weist eine typische Gatherer-Broker-Struktur auf. In der Gatherer-Phase des Suchprozesses wird aus den analysierten Internet-Informationen ein Suchindex gebildet, sowie die Erzeugung der *abstracts* vorgenommen. Im Broker-Prozeß werden die Nutzeranfragen durch das Dialogsystem auf Anfragen der IRQL[2] (Information Retrieval Query Language) abgebildet. Sie ermöglicht eine Kombination von Datenbank- und Information-Retrieval-Anfragen.

Natürlich-sprachliche Verarbeitung: Die natürlich-sprachliche Verarbeitung wird zum einen zur Analyse der Nutzeranfragen innerhalb des Dialogsystems verwendet. Zum anderen bildet sie die linguistische Grundlage zur Extraktion von Informationen aus den *abstracts*, sowie der Generierung natürlich-sprachlicher Antworten aufbauend auf den Fakten der *abstract*-Datenbank.

Ontologie: Die Ontologie repräsentiert ein semantisches Referenzmodell einer bestimmten Anwendungsdomäne, wodurch eine inhaltliche Einordnung der Suchanfrage ermöglicht wird und zudem Kontextinformationen bereitgestellt werden können.

Die Aufgabe der Universität Rostock im Gesamtprojekt GETESS ist die Realisierung der Suchmaschine. Darin enthalten ist die Konzeption einer verteilten

[2]Die IRQL wird derzeit am Lehrstuhl Datenbank- und Informationssystem entwickelt. Sie soll als Basissprache die Lücke zwischen IR und DB-Anfragen schließen.

Datenbanklösung zur Verwaltung der *abstracts*, der Wissensbasis und der Lexika sowie die Entwicklung einer Sprache für den einheitlichen Zugriff auf strukturierte und unstrukturierte Informationen (IRQL).

Die Untersuchungen der Möglichkeiten der Datentyperweiterungen, die im Rahmen dieser Diplomarbeit vorgenommen wurden, könnten hier den erster Ansatzpunkt hinsichtlich der IRQL in GETESS bilden.

1.2 Global-Info

Das Ziel des Förderkonzeptes "Globale Elektronische und Multimediale Informationssysteme" (Global-Info) des BMBF ist die Schaffung einer digitalen Bibliothek, die den Wissenschaftlern in Deutschland einen effizienten Zugang zu den weltweit vorhandenen elektronischen und multimedialen Informationen ermöglicht. Diese Informationen können in verteilten Informationssystemen digital gespeichert sein. Bei der Gestaltung der Strukturen wird insbesondere Wert auf das Zusammenwirken aller am Prozeß der Bereitstellung und Nutzung von Information und Dokumenten beteiligten Partner (Produzenten, Distributeure, Konsumenten) gelegt [Sch98].

Vom Global-Info Consortium wurden gemeinsame Grundsätze und die inhaltlichen Schwerpunkte für die Gestaltung des Projektes formuliert. Die fünf Schwerpunkte des Projektes sind nachfolgend aufgeführt.

1. Ergänzung und Bearbeitung von Inhalten: Dokumententypen, Verfahren und Werkzeuge für elektronisches Publizieren, Transfer, Speicherung, Konvertierung, Indizierung

2. die Vernetzung von Lehr- und Lernmaterialien

3. Formale Beschreibung, Identifikation und Retrieval, Metadaten, Vernetzung

4. Nutzung von Inhalten: Alterting, Awareness, Informationsverbände, Informationsvermittlung, Evaluierung von Ergebnissen, Oberflächen, Intelligente Agenten, Paßwortproblematik

5. Wirtschaftlichkeitsmodell, Billing und Abrechnung, Statistik

Die Einteilung des Projektes in die verschiedenen Schwerpunkte dient dabei nur der Gliederung und Organisation der Arbeit und nicht der inhaltlichen Abgrenzung. Vielmehr sind die Übergänge zwischen den Schwerpunkten fließend bzw. sie überlappen sich.

Die Beteiligung der Universität Rostock am Projekt Global-Info erfolgt über die Sonderfrodermaßnahme BlueRose, deren Zielsetzung und Einordnung in das Gesamtprojekt im folgenden Unterabschnitt erläutert wird.

1.2.1 Die Sonderfördermaßnahme BlueRose

Die Universität Rostock ist mit den Fachbereichen Informatik, Mathematik, Chemie und Physik sowie der Universitätsbibliothek und dem Rechenzentrum ein Projektpartner der Sonderfördermaßnahme BlueRose (*B*uilding *L*ibraries *U*nifying *E*nhanced *R*etrieval-*O*riented user *SE*rvices). Sie befaßt sich mit der Entwicklung von Retrieval-orientierten Diensten zur Nutzung elektronischer Literatur durch Wissenschaftler, Bibliothekare und Studenten. Die konkreten Ziele von BlueRose sind [Heu98]:

- eine einheitliche Benutzungsoberfläche für alle hier entwickelten Nutzer-Dienste
- die Vernetzung von (heterogenen) Dokumenten und anderen Informationen wie Tagungen, Autoren, Forschungsgruppen, Zeitschriftenreihen
- Integration verschiedener Techniken für Anfrage- und Retrieval-Funktionalitäten
- Bereitstellung aktueller, adaptierbarer, lokaler Sichten auf große verteilte Dokumentenbestände
- Benachrichtigung von Benutzern über Änderungen in den Fachinformationen im passiven und aktiven Fall
- Retrieval-Unterstützung durch fachspezifische, mehrsprachige Erschließungswerkzeuge

Aufgrund der Zielsetzung, ist BlueRose im wesentlichen dem Schwerpunkt 4 zuzuordnen. Abbildung 1.1 zeigt die Rahmenarchitektur des Projektes Global-Info. Farblich abgehoben ist dabei der Bereich, der durch die Zielsetzung von BlueRose abgedeckt wird.

Die Ziele, die im Rahmen von BlueRose durch die Universität Rostock verfolgt werden, sind zum einen die *Verwaltung semistrukturierter Datenbanken* zur Aufnahme von Volltexten und Metadaten über Dokumente. Diese Datenbanken sollen als lokale integrierte Dokumentenserver (LDS) in elektronischen Bibliotheken fungieren. Desweiteren wird durch die Integration von semistrukturierten Dokumenten und strukturierten Metadaten ein kombinierter Information-Retrieval und Anfrage-Prozeß impliziert. Dieser Prozeß erfordert geeignete Kopplungen für *verteilte Anfragen und verteiltes Retrieval.* Daneben sollen *Import-Funktionalitäten* für föderierte Datenbestände und Kopplungen des Anfrage- und

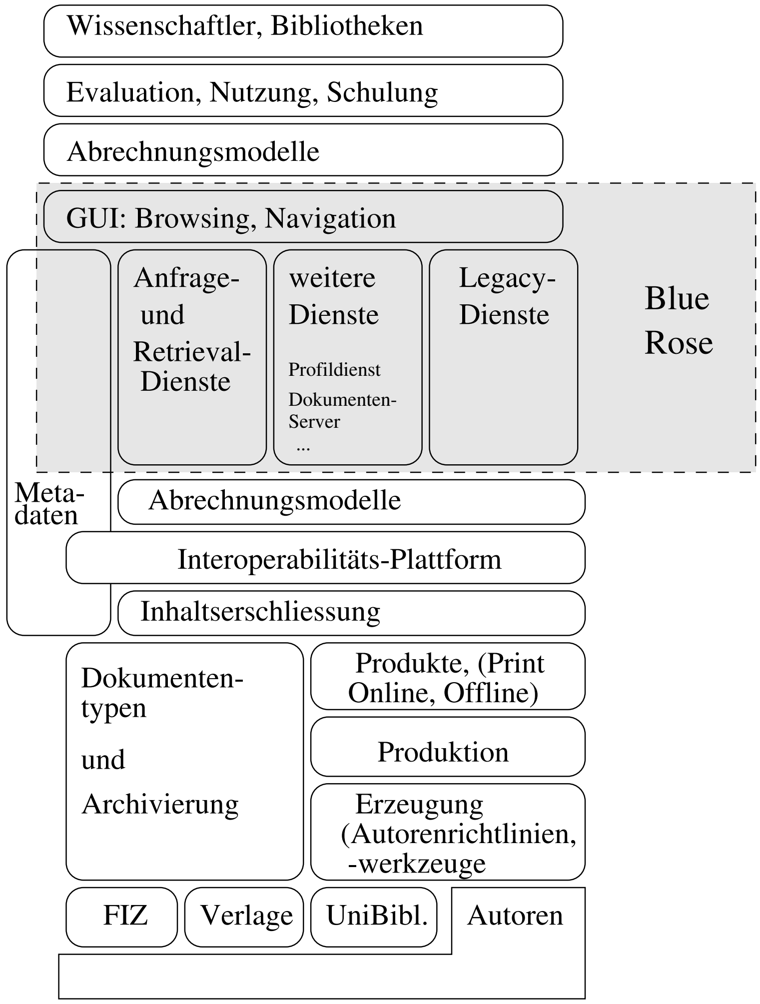

Abbildung 1.1: Einordnung von BlueRose in die Global-Info Rahmenarchitektur

Retrieval-Dienstes an Literatur-Recherche-Werkzeuge und Suchmaschinen wie beispielsweise MeDoc und InterDoc entwickelt werden.

Der Bezug zwischen den Untersuchungen zu Datentyperweiterungen im Rahmen dieser Arbeit und den Aufgaben der Universität Rostock im Global-Info Projekt wird durch das zu entwickelnde Modul des Lokalen Dokumenten Servers (siehe Abbildung 1.2) hergestellt. Dieses Modul dient der Aufnahme lokal zu verwaltender semistrukturierter Dokumente sowie strukturierter Informationen. Die zu speichernden Dokumente besitzen dabei eine implizite XML-Struktur (siehe Kapitel 3), die aufgrund ihrer Semantik eine gute Möglichkeit zur Integration von Datenbanken und Information-Retrieval bietet. Zur Realisierung des LDS wurde in BlueRose eine Kombination aus objektrelationalem und Volltext-

Datenbanksystem vorgeschlagen. Aufgrund der Erweiterbarkeit der objektrelationalen Datenbanksysteme wäre hier die Entwicklung eines XML-Datentyps auf Basis vorhandener Volltexterweiterungen denkbar.

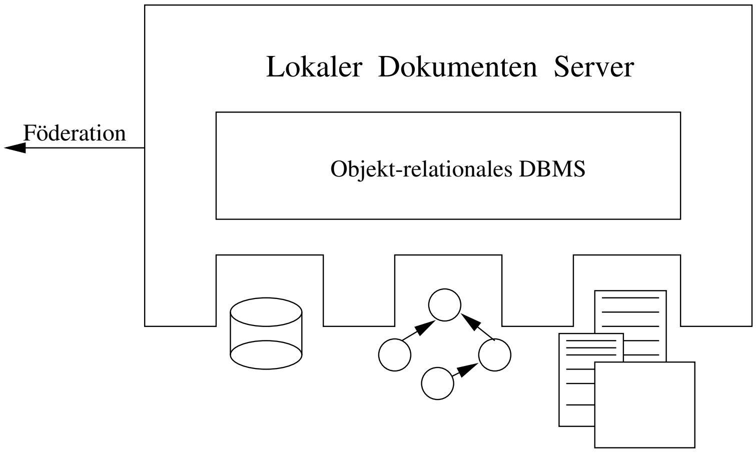

Abbildung 1.2: Lokaler Dokumenten Server (LDS)

1.3 Ziel der Arbeit

Aufbauend auf Untersuchungen der Volltext-Retrieval-Erweiterungen der Datenbanksysteme DB2 und Informix soll für beide Systeme eine Konzeption erarbeitet werden, die es ermöglicht, XML-Dokumente darzustellen, zu verwalten und mit Anfragemethoden finden zu können. Desweiteren wird für eines der beiden Systeme eine prototypische Implementierung vorgestellt. Ein weiteres Ziel der Diplomarbeit bildet der perspektivische Vorschlag eines XML-Datentyps, mit dem auf "ideale" Weise XML-Dokumente dargestellt, verwaltet und angefragt werden können.

1.4 Aufbau der Arbeit

Im weiteren Verlauf der Arbeit werden zunächst die Grundlagen zu *semistrukturierten Daten* und der Beschreibungssprache *XML* gegeben (Kapitel 2 und 3). Daran anschließend wird in Kapitel 4 die allgemeine Volltext-Retrieval-Funktionalität der Datenbanksysteme DB2 und Informix untersucht und einander gegenübergestellt. In diesem Kapitel soll sich zunächst nur die Leistungsfähigkeit im allgemeinen herauskristallisieren. Kapitel 5 steht unter der Fragestellung, inwieweit sich die Volltexterweiterungen der Datenbanksysteme für das

Information-Retrieval von semistrukturierten Dokumenten eignen und welche Erweiterungen auf diesen Systemen realisierbar sind. Für das Datenbanksystem DB2 wird neben der konzeptionellen Erweiterung die prototypische Realisierung vorgestellt. Da die Erweiterung eines bestehenden Volltext-Retrieval-Systems erwartungsgemäß nicht unproblematisch sein kann, wird in Kapitel 6 eine Konzeption eines "idealen" XML-Datentyps vorgestellt, der aufgrund seiner Funktionen gleichermaßen Anfragen über die Struktur als auch den Inhalt unterstützt. Zum Abschluß der Arbeit werden die wichtigsten Punkte noch einmal zusammengefaßt und ein Ausblick auf zukünftige Arbeiten gegeben.

Kapitel 2

Semistrukturierte Dokumente

Die Menge an elektronisch zur Verfügung stehenden Daten ist in den letzten Jahren dramatisch angestiegen. Die Palette reicht von unstrukturierten Daten in Dateisystemen zu hochstrukturierten Daten in Datenbanksystemen. Eine weitere Art, die man speziell im Zusammenhang mit dem World Wide Web antrifft, sind semistrukturierte Daten.

Im weiteren Verlauf diese Kapitels werden zunächst die Eigenschaften semistrukturierter Daten beschrieben sowie einige Datenmodelle für diese Dokumentenart vorgestellt (Abschnitt 2.1 und 2.2). Im Anschluß daran wird die Eignung dieser Datenmodelle hinsichtlich der Verwaltung und Anfrage von XML-Dokumenten aus dem Bereich "Digitale Bibliotheken" bewertet.

2.1 Eigenschaften semistrukturierter Daten

XML-Dokumente, deren Möglichkeiten zur Verwaltung und Anfrage in dieser Arbeit untersucht werden, fallen in die Kategorie semistrukturierter Dokumente. Aufgrund dessen, werden zunächst die charakteristischen Merkmale [Abi97] vorgestellt, durch die sich semistrukturierte Dokumente von anderen Dokumenten abheben.

- Die Struktur der Daten ist unregelmäßig

  Bei der Verwaltung einer Menge semistrukturierter Dokumente kann nicht davon ausgegangen werden, daß die Strukturelemente aller Dokumente homogen sind. Vielmehr können Elemente unvollständig sein oder zusätzliche Informationen beinhalten. Entsprechend kann auch die gleiche Information durch unterschiedliche Typen repräsentiert werden (Bsp. Preise in Euro oder Mark).

- Das Schema ist implizit in den Daten enthalten

  Semistrukturierte Dokumente besitzen in der Regel eine Struktur, die implizit enthalten ist.

  Die Möglichkeiten der Strukturinformationen und die zwischen den Strukturelementen bestehenden Beziehungen gehen über die Fähigkeiten von Standarddatenbankmodellen hinaus, so daß die Interpretation und Extrahierung der Struktur speziellen Applikationen beziehungsweise Werkzeugen obliegt.

- Die Struktur ist unvollständig

  Ein kaum lösbares Ziel ist die Strukturierung sämtlicher Daten, da diese vielfach einen sehr differenzierten Strukturierungsgrad aufweisen. Mit Hilfe von Information-Retrieval-Werkzeugen kann zwar eine eingeschränkte Strukturerkennung erfolgen, jedoch wird es immer Dokumente geben, die aus Datenbanksicht unstrukturiert sind.

- Das Schema ist flexibel

  In Standarddatenbanken wird eine strenge Typisierung angewendet. Jedes Attribut/Objekt des Datenbankschemas besitzt einen im voraus festgelegten Datentyp, wodurch nicht konforme Update-Operationen auf diesen Attributen/Objekten abgelehnt werden. Für viele Anwendungen ist diese strenge Typisierung jedoch zu einschränkend. Im Lore-Projekt [AQMW97] wird daher zur flexiblen Typisierung ein *data guide* verwendet. Dieser enthält zwar Informationen über den gegenwärtigen Typ; dieser muß jedoch nicht korrekt sein. Im Unterschied zu Update-Operationen einer Datenbank werden hier alle Daten akzeptiert.

- Das Schema ist relativ groß

  Aufgrund der Heterogenität der Dokumente ist das resultierende Schema, im Verhältnis zur Menge der Daten, relativ groß. Dies steht im Widerspruch zur Theorie relationaler Datenbanken, da hier davon ausgegangen wird, daß das Schema in Dimensionen kleiner ist als die zu speichernden Daten. Die Konsequenz ist, daß der Nutzer die Details der Struktur im allgemeinen nicht kennt und dadurch Anfragen über das Schema die gleiche Bedeutung erhalten, wie Anfragen über Daten.

- Das Schema unterliegt häufigen Änderungen

  In Standarddatenbanken wird davon ausgegangen, daß auf einem erzeugten Schema im weiteren Verlauf kaum Änderungen vollzogen werden, da diese in der Regel mit erheblichem Aufwand verbunden sind. Man betrachtet es daher auch als unveränderbar. Im Zusammenhang mit semistrukturierten Daten wird jedoch von einer sehr flexiblen Struktur ausgegangen, die häufigen Änderungen unterliegen kann. Diese Tatsache ist auch der Grund, warum bei der Verwaltung dieser Daten oft keine Datenbank verwendet wird.

- Der Unterschied zwischen Struktur und Daten ist unscharf

  In Standarddatenbanken wird grundsätzlich zwischen dem Schema (beschreibt die Struktur der Daten) und den Daten (stellen Instanzen dar) unterschieden. Im Zusammenhang mit semistrukturierten Daten verschwimmen diese Unterschiede. Hier können aufgrund der relativ häufigen Schemaänderungen Daten zu Schemainformation werden und umgekehrt Schemainformationen zu Daten.

Bei den im Zusammenhang mit dem Global-Info Projekt zu verwaltenden XML-Volltexten ist das Schema implizit im Dokument enthalten. Dieses kann aber in gewissen Grenzen, die durch die Document Type Definition (siehe Kapitel 3) festgelegt werden, variieren.

2.2 Datenmodelle für semistrukturierte Daten

Aufgrund der im letzten Abschnitt aufgeführten Merkmale eignen sich Datenbankmodelle im allgemeinen nicht zur Integration heterogener und wenig strukturierter Daten, da sie mangels Flexibilität ein festes und im voraus festgelegtes Schema erwarten. Treten Verletzungen dieses Schemas durch Update-Operationen auf, wird die Aufnahme dieser Daten in das Datenbanksystem abgelehnt. Diese Feststellungen waren der Grund für die Vorstellung neuer Integrationsmodelle, in denen weder Typen noch Schemata existieren.

2.2.1 Schemalose Datenmodelle

Der schemalose Ansatz ist ein populärer Weg zur Modellierung semistrukturierter Daten. In diesen Modellen wird kein vorher festgelegtes Schema zur Repräsentation der Daten gefordert, wodurch die Integration heterogener, semistrukturierter Dokumente möglich wird. Die benötigte Flexibilität der Modelle wird durch die Darstellung der Daten als Bäume oder Graphen erreicht, in die neue Daten durch Hinzufügen eines Knotens integriert werden können. Nachfolgend werden zwei Vertreter schemaloser Datenmodelle vorgestellt.

Object Exchange Model (OEM)

Daten im Object Exchange Model ([AQMW97]) werden durch einen Graphen $G = (V,E)$ repräsentiert, wobei E die Menge der Kanten und V die Menge der Knoten ist. Jede Kante des Graphen ist durch ein Label vom atomaren Typ *string* gekennzeichnet. Die Knoten des Graphen enthalten Objekte. Jedes Objekt besitzt eine eindeutige Objektidentität vom Typ *oid*. Alle Objekte des Graphen sind entweder atomar oder komplex. Werte eines atomaren Objektes müssen von einem atomaren Datentyp (*integer, real, string etc.*) sein. Der Wert eines komplexen Objektes ist die Menge der Objektreferenzen die von diesem Objekt ausgehen (bezeichnet durch die Menge der *(label, oid)* Paare). Abbildung 2.1 zeigt ein Beispiel eines OEM-Graphen, der mit Hilfe der Vorschriften der Buch-DTD[1] (siehe Anhang A.1) gebildet wurde. Die Buch-DTD legt eine gewisse Struktur der Dokumente fest, die jedoch zuviel Flexibilität für eine vernünftige Abbildung auf ein Datenbankschema bietet. Durch die Heterogenität können Strukturelemente semistrukturierter Dokumente unvollständig sein. Dies zeigt sich unter anderem in der Titelstruktur, die im Objekt &13 mit Ober- und Untertitel (Objekte &14 und &77), im Objekt &36 hingegen nur mit einem Obertitel (Objekt &49) angegeben wurde. Die Heterogenität der beiden Buch-Objekte wird jedoch deutlicher durch die unterschiedliche Struktur der beiden Body-Objekte (Objekte &23 und &11) zum Ausdruck gebracht.

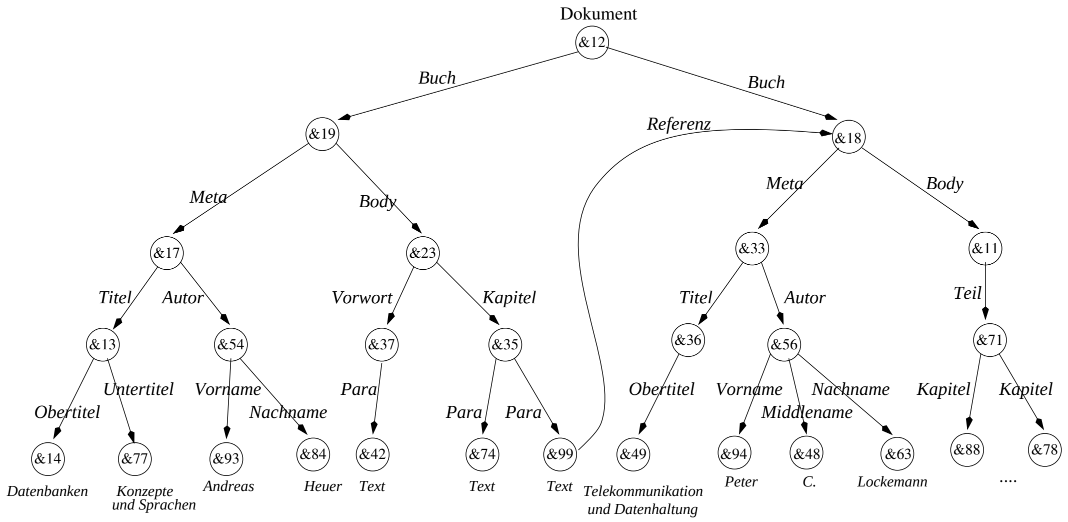

Abbildung 2.1: Beispiel eines OEM-Graphen

Das Baum-Modell

Das hier vorgestellte Baum-Datenmodell entstammt [BDS95]. In der vereinfachten Form beschreibt das Modell einen Baum mit markierten Kanten. Die Beschreibung der Kanten ist vom Typ *label*, der eine Vereinigung verschiedener Basis-Datentypen darstellt. Ein markierter Baum dieses Modells wird dann durch die rekursive Gleichung

[1] Aus Platzgründen wurden nur einige Elemente ausgewählt. Die Begriffserklärung für eine DTD wird in Kapitel 3 abgegeben.

$$tree = \mathcal{P}_{fin}\ (label \times tree)$$

repräsentiert, wobei $\mathcal{P}_{fin}(S)$ eine endliche Untermenge von S beschreibt.

Die Erstellung eines Baumes erfolgt mit Hilfe der folgenden Konstruktoren:

$$\begin{array}{lll} \phi & : & tree \\ \{\_\Rightarrow\_\} & : & label \times tree \rightarrow tree \\ \_\cup\_ & : & tree \times tree \rightarrow tree \end{array}$$

Bäume des Modells sind Mengen, so daß ein leerer Baum durch die leere Menge (ϕ) dargestellt wird. Der Ausdruck {l⇒ t} beschreibt einen Baum, dessen Wurzel eine mit l markierte Kante in Richtung des Unterbaumes t besitzt. Die dritte Konstruktionsregel erzeugt durch die Verschmelzung der Wurzeln der Bäume der rechten und linken Seite einen neuen Baum. In Abbildung 2.2 wird an Hand eines Beispiels die Vereinigung (c) durch das Verschmelzen der Wurzeln der Teilbäume (a,b) gezeigt.

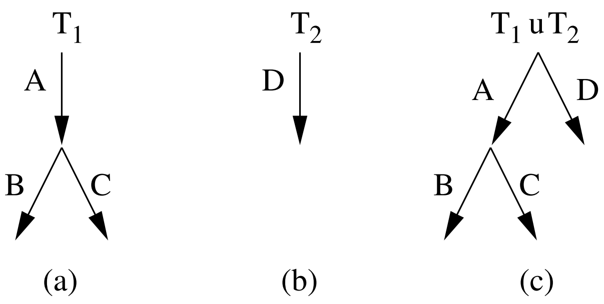

Abbildung 2.2: Beispiel der Vereinigung zweier Teilbäume

In [BDS95] wurde neben dem einfachen Baummodell eine Erweiterung vorgestellt, die auch Zyklen zwischen den Kanten des Baumes zuläßt (*rooted edge-labeled graph*).

Vorteile der schemalosen Modelle: Der Hauptvorteil dieser Modelle ist ihre Einfachheit und ihre Flexibilität.

Nachteile der schemalosen Modelle: Anfragen müssen dynamisch ausgeführt werden. In traditionellen Datenbanken wird eine zweistufige Anfrageverarbeitung durchgeführt. Zunächst wird mit Hilfe der Schemainformationen ein Ausführungsplan generiert. In der zweiten Phase wird ohne zusätzlichen Zugriff auf die Schemainformationen der Plan ausgeführt. In einem Modell ohne Schema ist dies nicht realisierbar. Hier müssen sämtliche Informationen dynamisch gefunden werden. Im schlechtesten Fall ist dazu eine Untersuchung der gesamten Datenbank erforderlich.

2.2.2 Das Union-Datenmodell

Der Ansatz der Union-Typen entstammt [Clu97]. Um die Flexibilität zu erreichen, die für die Verwaltung semistrukturierter Daten erforderlich ist, werden hier Union-Datentypen einem objekt-orientierten Datenbanksystem hinzugefügt. Die Frage der Typsicherheit bei Verwendung von Unions wird in dem Modell durch Bezeichung der Unions erreicht, die dann einen expliziten Zugriff auf die instantiierten Komponenten des Union erfordern und dadurch eine statische Typprüfung erlauben.

Formal werden die Union-Typen durch die nachfolgenden Regeln beschrieben.

1. Sind $\tau_1, ..., \tau_n$ von einander verschiedene Typen und $a_1, ..., a_n$ von einander verschiedene Attributnamen, dann ist $(a_1 : \tau_1 + ... + a_n : \tau_n)$ ein Union-Typ.
2. $dom(a_1 : \tau_1 + ... + a_k : \tau_k) = \{[a_i : v_i] | 1 \leq i \leq k \wedge v_i \in dom(\tau_i)\}$.
3. - if $\forall i \in [1..n], \exists j \in [1..m] : a_i = b_j \wedge \tau_i \leq \tau_j$
     $(a_1 : \tau_1 + ... + a_n : \tau_n) \leq (b_1 : \tau'_1 + ... + b_m : \tau'_m)$
   - if $\exists j \in [1..m] : a_i = b_j \wedge \tau_i \leq \tau_j$
     $[a_i : \tau_i] \leq (b_1 : \tau'_1 + ... + b_m : \tau'_m)$

Das Symbol zwischen den Typen $\leq$ repräsentiert eine Untertypbeziehung. Aufgrund dessen besagt Regel 3 beispielsweise, daß das Tupel $[a_i : \tau_i]$ einen Untertyp zu $(b_1 : \tau'_1 + ... + b_m : \tau'_m)$ darstellt, falls im Union-Typ ein Attributname existiert, der dem Attributnamen des Tupels entspricht und deren korrespondierende Typen in einer Ober-Untertypbeziehung stehen.

Die Verwendung der Union-Typen soll anhand des nachfolgenden Beispiels verdeutlicht werden.

Beispiel:

```
...
<!ELEMENT Body ( (Vorwort, (teil | kapitel+))
               | (teil | kapitel+) ) >
...
```

Das Beispiel zeigt einen sehr kleinen Ausschnitt aus der Buch-DTD (siehe Anhang A.1). Die Struktur des *Body*-Elementes beinhaltet entweder:

1. ein *Vorwort* gefolgt von einem *teil* oder einem oder mehreren *kapiteln*

oder

2. ein *teil* oder ein oder mehrere *kapitel*.

Diese verschiedenen Alternativen können mit Hilfe der eingeführten Union-Typen folgendermaßen dargestellt werden.

union (a_1 : tuple (vorwort : string,
union (teil : Teil, kapitel : list(Kapitel)))
a_2 : union (teil : Teil, kapitel : list(Kapitel)))

Werden innerhalb einer Anfrage Variablen verwendet, die auf einem Union-Typen operieren, wird eine implizite Selektion der entsprechenden Komponente des Union-Typen durchgeführt.

Vorteile des Modells: Der Hauptvorteil dieses Ansatzes ist, daß die Implementierung nur wenige Modifikationen des Datenbanksystems erfordert

Nachteile des Modells: Es ist nicht gesichert, ob Union-Typen die Lösung für alle Probleme der Integration heterogener Daten darstellt. Da vor jedem Zugriff überprüft werden muß, welche Komponenten des Union instantiiert sind, kann die intensive Nutzung der Unions zu einem ernstzunehmenden Leitungsabfall des Systems führen.

2.3 Bewertung und Wahl eines Modells

In diesem Abschnitt soll die Frage geklärt werden, durch welches Datenmodell die Strukturen und Operationen der im Rahmen dieser Arbeit zu verwaltenden semistrukturierten Dokumente sinnvoll repräsentiert werden können. Bei der Beurteilung sind dabei nicht nur die Vor- und Nachteile der betrachteten Modelle heranzuziehen, sondern auch, ob das Datenmodell auf das konkret zum Einsatz kommende Datenbanksystem abgebildet werden kann. Durch die Aufgabenstellung sind bereits Richtlinien hinsichtlich des zu verwendenden Datenbanksystems gesetzt. Sowohl Informix Universal Server als auch DB2 Universal Database stellen objektrelationale Datenbanksysteme dar.

1. **Object Exchange Modell:** Die Abbildung dieses Modells auf ein objektrelationales Modell ist durchführbar, womit die benötigte Flexibilität hinsichtlich der Integration heterogener Dokumente erreicht wird. Jedoch sind spätere Rekonstruktionen von Dokumenten und Browsing-Anfragen auf den Strukturen des objektrelationalen Modells kaum realisierbar. Eine andere Möglichkeit ist die Erhaltung der semistrukturierten Dokumente in ihrem Originalzustand. Für Operationen auf den Dokumenten würden die Dokumente dann mit Hilfe von Wrappern in das Graphenmodell transformiert. Diese Variante kann aber zu erheblichen Leistungseinbußen führen, da für jede Operation zunächst eine Transformation durchzuführen ist.

2. **Baum-Modell:** Die Ausführungen zum Object Exchange Modell treffen im wesentlichen auch auf das Baum-Modell zu.

3. **Union-Datenmodell:** Union-Typen, in der im Abschnitt 2.2.2 genannten Form sind in keinem der beiden objektrelationalen Modelle vorhanden. Sie könnten durch Nullwerte und Integritätsbedingungen simuliert werden. In Informix wäre auch die Modellierung durch eine Typ-Hierarchie möglich. Beide Varianten stellen aber keine adäquate Lösung dar. Zum einen bergen Nullwerte generell Probleme (beispielsweise werden bei Verbunden mögliche Tupel nicht in das Ergebnis aufgenommen), zum anderen können durch die Modellierung der Strukturen der DTD als Attribute von Relationen keine Browsing-Anfragen realisiert werden. Hierfür müßte die Struktur in einer Anfrage der Relationenalgebra vernachlässigt werden können.

Für das Datenmodell wird eine hybride Lösung vorgeschlagen, die zum einen aus strukturierten Anteilen, aber auch aus den semistrukturierten Daten bestehen wird. Da der Anwendungsbereich auf die Verwaltung und Anfrage semistrukturierter Daten aus dem Bereich "Digitale Bibliotheken" beschränkt ist, sind bestimmte Informationen über die Dokumente, auch aufgrund der Festlegungen der DTD, notwendig. Diese Informationen können durch spezielle Attribute des objektrelationalen Datenmodells repräsentiert werden. Die als semistrukturierte Dokumente vorliegenden Volltexte werden ebenfalls durch ein Attribut dargestellt, jedoch wird für Information-Retrieval-Anfragen eine Wrapper-Funktion eingesetzt, die eine Transformation der Dokumente von der Darstellung als Character Large Object (CLOB) in ein Baum-Modell durchführt. Durch die Transformation können Anfragen über die implizite Struktur und den Inhalt realisiert werden. Das im OEM-Modell angeführte Argument der Leistungseinbußen durch Wrapper-Einsatz tritt hier nicht in dem Maße auf, da nur IR-Anfragen auf Struktur und Inhalt eine Transformation erfordern.

Kapitel 3

XML - Extensible Markup Language

XML ist ein durch das World Wide Web Consortium (W3C) entwickeltes Dokumenten-Austauschformat. Trotz der oberflächlichen Ähnlichkeit zur Hypertext Markup Language (HTML) besteht zwischen beiden ein fundamentaler Unterschied. HTML ist eine spezielle Anwendung der Standard Generalized Markup Language (SGML [Gol90]), die zur Auszeichung von Dokumenten dient. XML stellt dagegen eine Teilmenge von SGML dar, die wie SGML nur die Vorschriften bereitstellt eine Auszeichnungssprache zu definieren. Die Notwendigkeit der Entwicklung einer Teilmenge von SGML bestand aufgrund der Komplexität, die das größte Hindernis von Anwendungen mit SGML darstellt.

Ein XML-basiertes Dokument besitzt sowohl eine logische als auch eine physische Struktur (siehe Abbildung 3.1). Die logische Struktur erlaubt eine Einteilung des Dokumentes in benannte Abschnitte und Unterabschnitte, sogenannte *Elemente*. Durch die physische Struktur kann ein XML-Dokument aus einer oder mehreren Speicherungseinheiten, sogenannten *Entities*, bestehen. Dadurch können Informationen wiederverwendet und Nicht-XML-Daten (wie z.B. Bilder) durch Referenzen in das Dokument eingefügt werden.

Im Kontrast zu HTML beschreibt XML nicht zwingendermaßen das Layout eines Dokumentes wohl aber dessen Verhalten. Somit handelt es sich um eine Metasprache, deren Ziel die Beschreibung anderer Sprachen ist. Es existieren keine vordefinierten Elemente, so daß die Benennung und Verwendung sich nach den Vorstellungen des Entwicklers richtet. Um jedoch eine möglichst gute Kontrolle über die logische Dokumentenstruktur zu gewährleisten, existiert ein optionaler Mechanismus, der es erlaubt, Elemente in einer speziellen Dokumentenklasse zu spezifizieren (Document Type Definition - DTD).

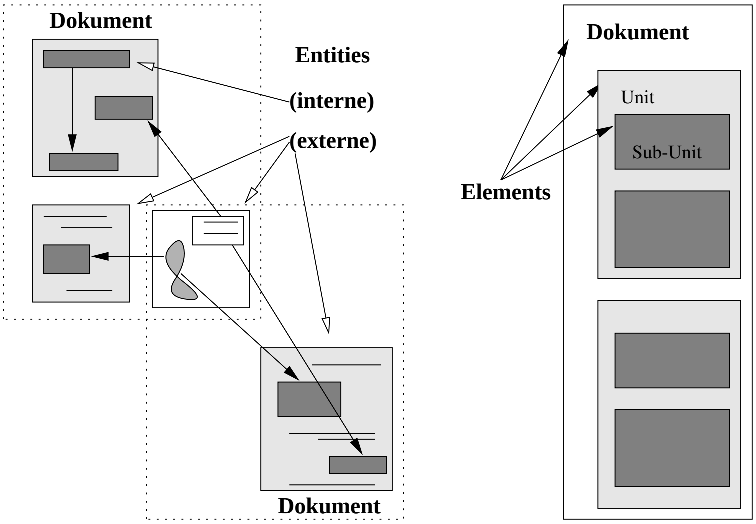

Abbildung 3.1: Physische vs. logische XML-Struktur [Bra98]

3.1 Document Type Definition (DTD)

Eine DTD ist durch eine formale Grammatik gekennzeichnet, die eine bestimmte XML-Sprache definiert. In ihr werden die Namen der Elemente (Tags) und ihre Schachtelung definiert (logische Struktur). Die Schachtelung wird erreicht, indem zu jedem Element angegeben wird, welche weiteren Elemente in seinem Inhalt auftreten dürfen. Die Strukturierung der Elemente kann durch nachfolgende logische Kombination von Elementen erreicht werden, die der Notation regulärer Ausdrücke ähnelt.

Sequenz:
(A , B) — *A* und *B* müssen in der angegebenen Reihenfolge aufgeführt werden

Alternative:
(A | B) — Hier muß entweder *A* oder *B* angegeben werden.

Wiederholung:
A* — A kann nullmal oder beliebig oft nacheinander angegeben werden.
A+ — A muß mindestens einmal angegeben werden.
A? — A kann nullmal oder genau einmal angegeben werden.

Jede DTD enthält eine Menge von Deklarationen, die eine Dokumenten-Baumstruktur definieren. Dieser Baum steht in einer bestimmten Beziehung zu allen Dokumenten (Instanzen), die dieser DTD entsprechen. Neben Elementen werden in einer DTD Attribute spezifiziert, die die dazugehörigen Elemente näher beschreiben. Beispiel 3.1 zeigt einen Ausschnitt aus der Diplomarbeit-DTD (siehe Anhang A.2).

Beispiel 3.1

```
<!–- Diplomarbeit DTD –->
<!ELEMENT Diplomarbeit (Titel, Bearbeitungszeit?)>
<!ATTLIST Diplomarbeit Sprache xml:lang NMTOKEN “de“>

<!ELEMENT Titel              (Obertitel, Untertitel?)>
<!ELEMENT Obertitel          (#PCDATA)>
<!ELEMENT Untertitel         (#PCDATA)>
<!ELEMENT Bearbeitungszeit   (Vom, Bis)>
<!ELEMENT Vom                (#PCDATA)>
<!ELEMENT Bis                (#PCDATA)>
```

Die Deklaration der einzelnen Elemente der DTD beginnt mit `<!ELEMENT` gefolgt von einem eindeutigen Elementnamen. Anschließend wird die Grammatik des Elementes entsprechend der zuvor aufgeführten logischen Kombination notiert. Attribute eines Elementes werden gesondert beginnend mit `<!ATTLIST` deklariert. Die Beispiel-DTD besteht somit aus den Elementen *Diplomarbeit, Titel, Obertitel, Untertitel, Bearbeitungszeit, Vom* und *Bis*. Das Element *Diplomarbeit* stellt das Wurzelelement dieser DTD dar. Seine innere Struktur wird durch die Sequenz aus *Titel* und *Bearbeitungszeit* vorgegeben, wobei die Angabe der Bearbeitungszeit optional ist. Zusätzlich wurde für dieses Element ein Attribut spezifiziert, welches die verwendete Sprache festlegt. Die Angabe von #PCDATA in der Deklaration eines Elementes verweist darauf, daß der Inhalt ausschließlich Text (*Parsed Character Data*) enthalten darf. Durch die XML-Spezifiktion [BPSM98] wird kein Typkonzept festgelegt, so daß Daten eines XML-Dokumentes grundsätzlich *PCDATA* oder *CDATA* (Character Data) sind.

3.2 Konformität

Bei der Prüfung der Korrektheit eines XML-Dokumentes wird zwischen den Begriffen *Wohlgeformtheit* und *Gültigkeit* unterschieden. Ein wohlgeformtes Dokument enthält mindestens ein oder mehrere Elemente, die durch ein Start- und End-Tag-Paar begrenzt werden und korrekt ineinander verschachtelt sind. Weiterhin existiert genau ein Wurzelelement, von dem kein Teil im Inhalt eines anderen Elementes enthalten ist. Als Konsequenz daraus gilt, daß für jedes Element k, das nicht die Wurzel ist, ein anderes Element v existiert, so daß sich k im Inhalt von v befindet, aber nicht im Inhalt eines anderen Elementes, das sich im Inhalt von v befindet.

1. v heißt Vater von k.
2. k heißt Kind von v.

Desweiteren darf in einem wohlgeformten Dokument kein Attributname mehr als einmal in einem Element verwendet werden und der Attributwert darf keinen Verweis auf ein externes Entity oder eine öffnende spitze Klammer enthalten.

Ein gültiges XML-Dokument hingegen darf die logische Struktur seiner DTD nicht verletzen. Das bedeutet, daß ein gültiges Dokument keine Verletzungen der in [BPSM98] enthaltenen Gültigkeitsbeschränkungen aufweisen darf. Jedes gültige Dokument ist auch wohlgeformt. Beispiel 3.2 zeigt eine gültige und damit auch wohlgeformte Instanz der DTD aus Beispiel 3.1. Es ist insofern gültig, als daß der Name der Dokumenttyp-Deklaration (DOCTYPE) mit dem Namen des Wurzelelementes korrespondiert und keine Gültigkeitsbeschränkung der XML-Spezifikation durch die Instanz verletzt wird.

Beispiel 3.2

```
<!DOCTYPE Diplomarbeit SYSTEM "da.dtd">
<Diplomarbeit Sprache="de">
  <Titel>
    <Obertitel>
      Untersuchungen zu Datentyperweiterungen für XML-Dokumente und
      ihre Anfragemethoden am Beispiel von DB2 und Informix
    </Obertitel>
  </Titel>
  <Bearbeitungszeit>
    <Vom>
      01.11.1998
    </Vom>
    <Bis>
      30.04.1999
    </Bis>
  </Bearbeitungszeit>
</Diplomarbeit>
```

Die Korrektheit von Dokumenten wird mit Hilfe von XML-Prozessoren überprüft. Konforme Prozessoren werden in *validierend* und *nicht-validierend* kategorisiert, wobei in Abhängigkeit dieser Kategorie auf Gültigkeit oder Wohlgeformtheit getestet wird.

Validierende Prozessoren müssen Verletzungen der Beschränkungen, die durch Deklarationen in der DTD formuliert werden, sowie jedes Nicht-Erfüllen der in der XML-Spezifikation formulierten Gültigkeitsbeschränkungen melden. Zur Erreichung dieses Ziels müssen sie die gesamte DTD und alle extern analysierten Entities, die im Dokument referenziert werden, einlesen und verarbeiten.

Nicht-validierende Prozessoren müssen lediglich ein Dokument einschließlich der internen DTD-Teilmenge auf Wohlgeformtheit prüfen. Dazu müssen sie alle Deklarationen einer internen DTD-Teilmenge und interne Parameter-Entities lesen und verarbeiten.

Kapitel 4

Evaluierung der Retrieval-Funktionalität von DB2 und Informix

In der Einführung dieser Arbeit wurde bereits auf die zunehmende Bedeutung von Datenbanken zur Speicherung, Verwaltung und Manipulation von multimedialen Dokumenten hingewiesen. Diese Datenbanken stellen jedoch neben der Forderung, große Datenmengen (CLOB[1], BLOB[2]) speichern zu können auch neue Herausforderungen an das Anfragesystem, da hier die strukturierten Anfragemöglichkeiten zur gezielten Informationsextraktion auf diesen Datenmengen versagen. Eine getrennte Anfrage zwischen den strukturierten Datenbankanteilen und den unstrukturierten Datenmengen über ein externes Retrieval-Werkzeug ist jedoch inakzeptabel. Einige Datenbankhersteller haben diesem Problem bereits Rechnung getragen und ihre Datenbanksysteme dahingehend erweitert, daß es möglich ist, neue Datentypen und Funktionen in das System zu integrieren, die Aufgaben ausführen, die zuvor durch externe Werkzeuge realisiert wurden. Durch diese Erweiterungen sind kombinierte Anfragen aus strukturierten und unstrukturierten Datenbankanteilen realisierbar.

Der überwiegende Teil diese Kapitels widmet sich den beiden objektrelationalen Datenbanksystemen DB2 und Informix und im speziellen ihren Volltext-Erweiterungen. Diese Erweiterungen, basierend auf neuen Datentypen und zugehörigen Funktionen, sind integraler Bestandteil der Datenbanksysteme und bieten somit die Möglichkeit der kombinierten Anfrage über strukturierte und unstrukturierte Datenbankanteile.

Im weiteren Verlauf wird zunächst eine kurze Einführung in die wichtigsten Begriffe des Information-Retrieval gegeben. Anschließend wird sowohl für das

[1] CLOB - Character Large Object
[2] BLOB - Binary Large Object

Datenbanksystem DB2 als auch für Informix die Funktionalität der jeweiligen Volltext-Retrieval-Erweiterung eingehend vorgestellt. Den Abschluß dieses Kapitels bildet dann eine Gegenüberstellung der beiden Systeme hinsichtlich ihrer Retrieval-Leistungsfähigkeit.

4.1 Begriffsklärung

Retrieval-Modelle: Information-Retrieval befaßt sich mit den Konzepten der Repräsentation und der Interpretation der Struktur von Informationen. Information-Retrieval selbst ist ein Kommunikationsprozeß, der auf der Eingabeseite aus der Analyse von Daten durch Verfahren der Wissensrepräsentation und deren Überführung in gespeichertes Wissen besteht und auf der anderen Seite Informationen liefert, die aufgrund von Transformationen aus der internen Wissensstruktur gewonnen werden.

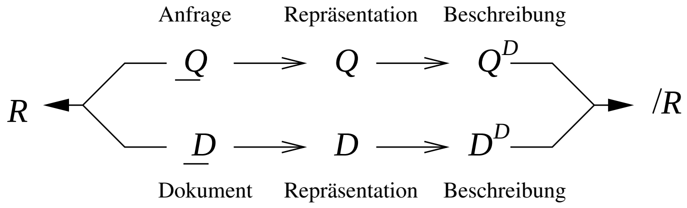

Abbildung 4.1: Konzeptionelles Modell für Retrieval

[Fuh97] verwendet das in Abbildung 4.1 dargestellte Modell als grundlegendes konzeptionelles Modell für Information-Retrieval. Dabei bezeichnet $\underline{D}$ die Menge der Dokumente in der Datenbasis und $\underline{Q}$ die Menge der Anfragen an das Retrieval-System. Zwischen den Dokumenten und den Anfragen besteht die Relevanzbeziehung R. Die Dokumentenrepräsentation D ist die semantische Sicht auf Dokumente. Die formalisierten Anfragen werden als Fragerepräsentation Q bezeichnet. Zum Zwecke des Retrievals werden die Repräsentationen in Beschreibungen (D^D, Q^D) überführt. Fragebeschreibungen sind beispielsweise boolesche Ausdrücke, Dokumentenbeschreibungen können gewichtete oder ungewichtete Indexierungen sein. Durch eine Retrieval-Funktion wird die Beschreibung von Frage-Dokument-Paaren verglichen und daraus ein Retrieval-Gewicht berechnet (im allgemeinen eine reelle Zahl). Die Erstellung der Beschreibungen und die Definition einer Retrieval-Funktion hängt vom jeweiligen Retrieval-Modell ab. Die nachfolgenden Unterpunkte geben eine kurze Beschreibung der wichtigsten Retrieval-Modelle[3].

[3] Es existieren weitere Retrieval-Modelle wie beispielsweise das Probabelistische Modell, die aber nur theoretischen Charakter besitzen.

Boolean-Retrieval: Boolean-Retrieval ist das bekannteste und historisch erste Retrieval-Modell. Der Anwender formuliert seinen Informationsbedarf als eine logische Kombination von Schlüsselworten, die durch die logischen Operatoren *AND*, *OR* und *NOT* verbunden werden. Aufgrund der binären Gewichtung der Terme kann die Retrieval-Funktion nur die Retrieval-Gewichte 0 und 1 liefern. Daraus resultiert eine Zweiteilung der Dokumente der Datenbasis in gefundene und nicht gefundene Dokumente.

- Vorteile des Modells:
  - effizient zu implementieren
  - Mächtigkeit — mit einer booleschen Anfrage kann jede beliebige Teilmenge von Dokumenten aus einer Datenbasis selektiert werden
- Nachteile des Modells:
  - Größe der Antwortmenge ist schwer zu kontrollieren
  - keine Ordnung der Antwortmenge nach mehr oder weniger relevanten Dokumenten, diese kann nur simuliert werden (beispielsweise durch Bestimmung der Vorkommenshäufigkeiten des Suchterms)
  - keine Gewichtung der Frageterme
  - strenge Trennung in gefundene und nicht gefundene Dokumente
  - schwierige Frageformulierung

Fuzzy-Retrieval: Zur Überwindung einiger Nachteile des booleschen Retrievals wurde das Fuzzy-Retrieval vorgeschlagen, dessen Grundlage die Theorie der Fuzzy-Logik bildet. Im Unterschied zum booleschen Retrieval können den Attributen gewichtete Indexierungen zugeordnet werden, wodurch die Retrieval-Funktion Werte im Intervall [0,1] liefert. Damit ergibt sich nun eine Rangordnung der Antwortdokumente.

- Vorteile des Modells:
  - Rangordnung der Antwortdokumente
- Nachteile des Modells:
  - keine Fragetermgewichtung
  - schwierige Frageformulierung
  - schlechte Retrieval-Qualität im Verhältnis zum Vektorraummodell

Vektorraummodell: Das Vektorraummodell wurde ursprünglich im Rahmen des SMART-Projektes [Sal71] entwickelt. Im Vektorraummodell werden Dokumente und Fragen als Punkte in einem Vektorraum aufgefaßt, der durch die Terme der Datenbasis aufgespannt wird. Beim

Retrieval wird dann nach solchen Dokumenten gesucht, deren korrespondierende Vektoren ähnlich dem Fragevektor sind. Für den Vektorraum wird dabei vorausgesetzt, daß alle Term-Vektoren orthogonal und normiert sind.

- Vorteile des Modells:
  - anschauliches Modell aufgrund der geometrischen Interpretation
  - benutzerfreundlich aufgrund einfacher Frageformulierung
  - Möglichkeit der Verbesserung der Retrieval-Qualität durch Relevance-Feedback
- Nachteile des Modells:
  - Modell enthält sehr viele Heuristiken[4]

Ranking: Ranking ist ein Rangordnungssystem, das jedem Dokument einen Meßwert zuordnet, der die Güte des Dokumentes zum Frageterm widerspiegelt. Für die oben beschriebenen Retrieval-Modelle hat das Ranking nur für Fuzzy-Retrieval und das Vektorraummodell Bedeutung. Aufgrund der strengen Trennung der Dokumente beim Boolean-Retrieval existiert hier keine Rangordnung. Diese könnte lediglich nachträglich durch Bestimmung der Häufigkeiten der Attributwerte erzeugt werden.

Relevance-Feedback: Bei der Begriffsbeschreibung des Vektorraummodells wurde als ein Vorteil gegenüber Boolean- und Fuzzy-Retrieval die Möglichkeit der Verbesserung der Retrieval-Qualität durch Relevance-Feedback genannt. Beim Relevance-Feedback möchte man durch Relevance-Beurteilung der qualitativ besten Dokumente einer Anfrage die ursprünglichen Fragetermgewichte so verändern, daß sich ein verbesserter Fragevektor ergibt. Das Retrieval kann dann mit dem verbesserten Fragevektor erneut durchgeführt werden. Dieser Prozeß (Relevance-Beurteilung, Retrieval) kann auch iterativ erfolgen.

Indexierung: Ein Information-Retrieval wird in der Regel auf sehr großen Datenbeständen durchgeführt. Die Indexierung der Dokumente eines IR-Systems ist daher ein wichtiger Faktor, der sich trotz des Mehraufwandes an Speicher und Update-Kosten positiv auf die Performance bei der Lokalisierung der relevanten Dokumente einer Suchanfrage auswirkt. In [Alm98] werden verschiedene Indizierungsformen vorgestellt. Einige dieser Indizierungsarten werden sich in den nachfolgend beschriebenen Retrieval-Erweiterungen der Datenbanksysteme wiederfinden.

[4]Nachteil des Modells ist, daß durch den heuristischen Ansatz zur Berechnung der Indexierungsgewichte die Dokumentenrepräsentationen nur schlecht erweitert werden können. Hierzu müssen erst geeignete Gewichtungsformeln durch Experimente gefunden werden.

4.2 IBM DB2 Universal Database

DB2 Universal Database (UDB) ist das derzeit jüngste Mitglied der DB2 Produktfamilie. Der Name Universal Database soll dabei die Vielfalt an unterstützten Systemplattformen und Einsatzbereichen signalisieren. Die nachfolgenden Punkte stellen einige der in [Cha98] genannten Eckpfeiler der UDB dar:

- Unterstützung unterschiedlichster Hard- und Software-Umgebungen
  UDB Server existieren für Windows NT, OS/2 und viele Unix-basierte Systeme, einschließlich AIX, Solaris und HP-UX. UDB Clients sind für Window95, Windows3.1 und Macintosh-Systeme lieferbar.
- Unterstützung zweier unabhängiger Arten von Parallelität (symmetrisches Multiprozessing und massiv parallele "shared nothing" Konfigurationen)
- Sprachunterstützung für C, C++, Java, FORTRAN, COBOL und REXX für die Einbettung von SQL-Statements in Applikationen
- Unterstützung von Industriestandards wie ODBC[5] und SQL92[6]
- Graphische Schnittstellen für die interaktive Datenbanknutzung und zur Datenbankadministration
- Bereitstellung von erweiterten Datentypen und zugehörigen Funktionen

In der Produktspezifikation klassifiziert IBM die UDB als objektrelationales Datenbanksystem. Die Datenbankfunktionalität stützt sich im wesentlichen auf relationale Konzepte, die durch die Möglichkeit der Integration nutzerdefinierter Datentypen[7] und Funktionen[8] erweitert wurden. Bei der Definition von neuen Datentypen besteht jedoch die Einschränkung, daß jeder neue Datentyp von einem Standard-Datentyp abgeleitet werden muß und nicht komplex strukturiert sein darf.

Neben der Option, entsprechend des Anwendungsfalles eigene Datentypen zu entwickeln, werden durch die UDB erweiterte Datentypen und Methoden für die verschiedenen multimedialen Bereiche (Audio, Video, Image, Text) bereitgestellt.

Im folgenden Abschnitt soll die Funktionalität der Erweiterung für den Bereich Text und Textrecherche eingehender betrachtet werden. Zu diesem Zweck wurden die im Produkthandbuch [IBM97b] beschriebenen Retrieval-Funktionalitäten durch diverse Testanfragen evaluiert. Insofern durch die Analyse des Systems Abweichungen vom beschriebene Verhalten in [IBM97b] registriert wurden, wird dies in den entsprechenden Unterpunkten zur Funktionsbeschreibung vermerkt.

[5] ODBC - Open Database Connectivity
[6] SQL - Structured Query Language
[7] UDT - user defined data types
[8] UDF - user defined functions

4.3 DB2 TextExtender

Der TextExtender in der Version 5.2 ist ein Mitglied der DB2 Extender-Familie. Er stellt eine Erweiterung dar, die es ermöglicht, innerhalb einer strukturierten SQL-Anfrage Operationen zur Volltextsuche zu spezifizieren. Durch diese Erweiterung kann ein kombinierter Information-Retrieval und Anfrage-Prozeß verwirklicht werden. Zusätzlich zu den nutzerdefinierten Funktionen enthält der TextExtender ein Application Programming Interface (API), dessen Funktionen in Anwendungsprogrammen zur Textsuche und -darstellung verwendet werden können.

Mit Hilfe der weiteren Mitglieder der Extender-Familie

- Image-Extender
- Audio-Extender
- Video-Extender

können kombinierte Suchanfragen über Image-, Audio- und Video-Datentypen in einer SQL-Anfrage realisiert werden.

Innerhalb dieses Abschnittes werden die verschiedenen Indizierungsformen sowie die Suchfunktionalität des TextExtenders beschrieben.

4.3.1 Indizierung

Obligatorisch für die Nutzung der Retrieval-Funktionalität des TextExtenders ist die Erzeugung eines Indexes auf den zu durchsuchenden Dokumenten, da die Funktionen nicht auf den Dokumenten selbst, sondern auf dem daraus gebildeten Index operieren. Jegliche Änderung an den Dokumenten sollte daher immer einen Index-Update nach sich ziehen, da andernfalls ein verfälschtes Retrieval-Ergebnis die Folge sein kann. Der TextExtender speichert die indizierten Terme[9] als invertierte Liste, das heißt, daß zu jedem Term die Dokumente vermerkt werden, die diesen Term enthalten.

Der TextExtender bietet verschiedene Indizierungsarten an, wobei durch jeden Index jeweils nur eine echte Teilmenge der zur Verfügung stehenden Suchoperationen unterstützt wird. Da pro Dokument/Attribut jedoch nur eine Indizierungsform zulässig ist, sind im vorhinein Überlegungen anzustellen, welche Suchoperationen primär genutzt werden sollen.

[9]Die indizierten Terme entsprechen je nach Art des Index entweder der exakten Form oder der Stammform des Begriffes im Dokument.

An dieser Stelle werden zunächst nur die Unterscheidungsmerkmale der Indizes aufgelistet. Die auf den jeweiligen Indizes möglichen Suchoperationen werden in Abschnitt 4.3.3 tabellarisch aufgezeigt.

Linguistischer Index: Die Erzeugung des linguistischen Indexes erfordert eine linguistische Verarbeitung des Textes. Folgende Schritte werden dabei vollzogen, wobei einige Funktionen in Abhängigkeit von der Dokumentensprache entfallen (die Reihenfolge der Schritte impliziert nicht die Reihenfolge ihrer Ausführung):

- Wort- und Satztrennung
- Erkennung der Satzanfänge
- Aufhebung von Worttrennungen
- Wortnormalisierung — Umformung von Großbuchstaben in entsprechende Kleinbuchstaben und Ersetzung von akzentuierten Buchstaben durch nicht akzentuierte Formen
  Bsp.: Tür $\longrightarrow$ tuer
- Stammformreduktion
- Dekomposition zusammengesetzter Wörter (nur deutschsprachig)
- Stoppwortfilterung
- Part-of-speech Filterung
  Diese Wortfilterung ist ähnlich der Stoppwortfilterung. Es werden nur Substantive, Verben und Adjektive indiziert.
  Bsp.: I drive my car quickly. Indiziert werden *car* und *drive*. Die Worte *I* und *my* werden als Stoppworte erkannt, zusätzlich wird aber auch *quickly* durch die Part-of-speech Filterung aus der Indizierung entfernt.
- Feature extraction — Erkennung von Namen, Eigennamen, Abkürzungen
  Feature extraction wird nur für englischsprachige Dokumente unterstützt. Diese Option muß explizit bei der Indexerzeugung spezifiziert werden.

Bei einer Anfrage gegen einen linguistischen Index wird vor der Suche im Index die gleiche linguistische Verarbeitung auf die Begriffe des Suchterms angewendet.

Präziser Index: In dieser Indizierungsform werden die Begriffe entsprechend ihres Auftretens im Dokument indiziert. Der Prozeß der Indexerstellung umfaßt dann nur die Schritte

- Wort- und Satztrennung

- Stoppwortfilterung

Dualer Index: Der duale Index stellt eine Kombination aus linguistischem und präzisem Index dar, so daß innerhalb eines Dokumentes sowohl nach der exakten als auch nach der Stammform gesucht werden kann. Die größere Suchfunktionalität schlägt sich jedoch in der Größe des Index und damit in einer längeren Antwortzeit nieder.

Ngram Index: Im Unterschied zu den zuvor beschriebenen Indizes, basiert der Ngram Index nicht auf einem Wörterbuch. Dem Namen des Index nach zu urteilen, beruht er auf dem n-gramm Ähnlichkeitsmaß, das die orthographische Ähnlichkeit zwischen zwei Zeichenketten dadurch berechnet, daß es die Anzahl der übereinstimmenden Buchstabenfolgen (n-Gramme) bestimmt und zur Gesamtzahl aller in beiden Zeichenketten vorkommenden n-Gramme ins Verhältnis setzt. Dies ist die einzige Indizierungsform, die die Fuzzy-Suche unterstützt, so daß Zeichenketten gefunden werden können, die ähnlich dem spezifizierten Suchterm sind.

Enthalten die zu indizierenden Dokumente DBCS-Zeichen[10] (chinesische, japanische, koreanische Schriftzeichen), so stellt der Ngram Index die einzig mögliche Indizierungsform dar, da DBCS Zeichen durch keinen anderen Index unterstützt werden.

Soll mit Hilfe des Ngram Index eine exakte Suche vorgenommen werden, so ist bei der Indexerzeugung der Parameter *case_enabled* anzugeben.

Für das Erzeugen, Ändern und Löschen von Indizes zur Nutzung der Retrieval-Funktionalität steht ein Kommandozeilenprozessor zur Verfügung, der nicht Bestandteil des DB2 Datenbanksystems ist. Die Speicherung der Indexdateien erfolgt ebenfalls außerhalb der Datenbankumgebung, so daß an dieser Stelle auch nicht die Sicherheitsmechanismen des Datenbanksystems greifen. Dies stellt nur in Hinsicht auf die relativ gute Lesbarkeit der Datendateien des Index ein Sicherheitsrisiko dar, da hierdurch Informationen aus den Dokumenten unbefugt gelesen werden können. Die Verbindung zwischen den externen Indexdateien und dem internen Aufruf der Retrieval-Funktion wird durch die Erzeugung eines Handle-Attributs in der Dokumentenrelation geschaffen. Da sich auch das Dokument selbst außerhalb des Datenbanksystems befinden kann, besteht diese Relation u.U. nur aus dem Handle-Attribut selbst.

4.3.2 Suchfunktionalität

Durch den TextExtender wird, unabhängig ob die Suche interaktiv oder aus einem Anwendungsprogramm erfolgt, zur Beantwortung von Information-Retrieval-An-

[10]DBCS - double byte character sets

fragen lediglich das Boolean-Retrieval-Modell unterstützt.

Für die interaktive Suche mit Hilfe des TextExtenders stehen die folgenden vier UDF bereit.

CONTAINS: Diese Funktion stellt die wichtigste Funktion bei der Volltextsuche mit Hilfe des TextExtenders dar. Das Ergebnis des Funktionsaufrufs sind die booleschen Werte *true* (1) oder *false* (0).

$$Suchergebnis \begin{cases} 1 & : \quad Suchargument \ \ tritt \ \ im \ \ Dokument \ \ auf \\ 0 & : \quad sonst \end{cases}$$

Bei der CONTAINS-Funktion handelt es sich um eine skalare Funktion, die nicht in der FROM-Klausel eines SFW[11]-Blockes angegeben werden darf.

NO_OF_MATCHES: Diese Funktion liefert als Ergebnis die Anzahl der Treffer des Suchargumentes je Dokument. Es handelt sich hierbei ebenfalls um eine skalare Funktion, so daß dieselben Bedingungen gelten, wie für die CONTAINS-Funktion.

RANK: Die Ranking-Funktion soll die Güte eines Dokumentes in Bezug auf das Suchargument widerspiegeln. Beim TextExtender wird diese Rangfolge durch das Verhältnis der Anzahl der Treffer (NO_OF_MATCHES) zur Dokumentengröße errechnet. RANK ist ebenfalls eine skalare Funktion.

SEARCH_RESULT: Im Unterschied zu den zuvor aufgeführten Funktionen liefert diese UDF eine temporäre Relation, bestehend aus den drei Attributen NO_OF_MATCHES, RANK und Handle als Ergebnis des Aufrufs zurück, wobei nur Dokumente selektiert werden, die das spezifizierte Suchargument erfüllen. Der Vorteil dieser Funktion ist die signifikante Verbesserung des Durchsatzes bei der Anwendung auf sehr große Relationen im Vergleich zur RANK oder CONTAINS-Funktion. Der Aufruf der Tabellenfunktion ist nur innerhalb einer FROM-Klausel zulässig.

Jede dieser Funktionen sucht im Index nach dem Vorkommen des spezifizierten Sucharguments. Zur effizienteren Anfrageausführung wird das Vorhandensein des Sucharguments im Index nicht bei jedem Aufruf der Funktion geprüft. Statt dessen wird beim ersten Aufruf der UDF eine interne Liste erstellt, die alle Dokumente enthält, die das Suchargument erfüllen. Für jeden weiteren Aufruf auf der Menge der Dokumente einer Relation wird dann aufgrund der internen Liste bestimmt, ob das Dokument in die Ergebnismenge aufgenommen werden muß.

4.3.2.1 Boolesche Suche

Mit Hilfe der booleschen Operatoren & (AND), | (OR) und ! (NOT) wird eine logische Verknüpfung der Suchatome (Suchbegriff/Phrase + Attributierung) erreicht.

[11]SFW - Select From Where

Die Auswertungsreihenfolge ist dabei von links nach rechts, wobei das logische *AND* stärker bindet als das logische *OR*. Soll eine andere Auswertungsreihenfolge erzielt werden, müssen die einzelnen Gruppen durch Klammerung geschachtelt werden. Der logische *NOT* Operator hat in Bezug auf *AND* und *OR* eine etwas eingeschränkte Anwendungsmöglichkeit. So darf direkt nach *NOT*, entsprechend den Regeln der Suchalgebra, nur ein *SearchPrimary* folgen. Ein *SearchPrimary* wird dabei durch folgende EBNF[12]-Regeln beschrieben (siehe Anhang A):

SearchPrimary ::= SearchAtom |
'(' SearchPrimary1 ')'

SearchPrimary1 ::= SearchAtom |
',' SearchPrimary1

Das bedeutet, daß nach *NOT* entweder nur ein einzelnes Suchatom (Suchbegriff/Phrase + Attributierung) oder eine durch Komma getrennte Liste von Suchatomen, die durch Klammern begrenzt wird, folgen darf. Die durch Komma getrennten Suchatome werden entsprechend einer Disjunktion ausgewertet.

4.3.2.2 Exakte Suche

Soll die exakte Form eines Suchterms im Index gefunden werden, so kann dies mit Hilfe des *PRECISE FORM OF* Attributs erfolgen. Voraussetzung hierfür ist jedoch ein präziser, dualer oder Ngram Index. Letzterer muß zusätzlich mit der *case_enabled* Option erstellt worden sein.

4.3.2.3 Stammformsuche

Die Suche nach Variationen eines Terms wird durch die Stammformsuche ermöglicht (Attribut *STEMMED FORM OF*). Voraussetzung hierfür ist jedoch ein linguistischer oder dualer Index. Laut Produkthandbuch ist die Stammformsuche auch auf dem Ngram Index ausführbar. Da dieser Index aber nicht auf einem Wörterbuch basiert, stellt die Stammformsuche auf diesem Index nur eine Teilstringsuche dar.

Tests dieses Attributs haben leider gezeigt, daß der TextExtender speziell bei deutschsprachigen Dokumenten nicht immer das erwartete Resultat erzielt. So konnten z.B. bei einer Stammformsuche nach `"buch"`, Dokumente die den Begriff `"Buch"` enthielten nicht gefunden werden, weil laut Aussage der TextExtender Entwicklungsabteilung das Wort *buch* als Imperativ des Verbs *buchen* interpretiert und daher auf das Verb *buchen* abgebildet wird. Bei den Versuchsanfragen

[12]EBNF - erweiterte Backus-Naur Form

wurden noch weitere Begriffe (Titel, Abschnitt) gefunden, die in ihrer Kleinschreibung durch den TextExtender nicht gefunden wurden.

4.3.2.4 Fuzzy-Suche

Die Fuzzy-Suche ermöglicht es, Terme eines Dokumentes zu finden, die Transpositionen bzw. Substitutionen von Zeichen enthalten können. Wie groß die Ungenauigkeiten des Begriffes dabei sein dürfen, hängt von der im Suchausdruck angegebenen Stufe ab. Die Stufe eins läßt mit nur 20 % die größten Freiheiten bei der Treffergenauigkeit zu. In der Stufe fünf wird eine Genauigkeit von 90 % zwischen dem Suchterm und den Begriffen des Index gefordert. Die Fuzzy-Suche wird über das Attribut *FUZZY FORM OF* eingeleitet, wobei auf den Dokumenten ein Ngram Index vorhanden sein muß.

Für die Fuzzy-Suche gilt neben der Einschränkung auf den Ngram Index, daß mindestens die ersten drei Buchstaben des Wortes mit dem Suchterm übereinstimmen müssen, wodurch Suchbegriffe mit Rechtschreibfehlern am Wortanfang nicht lokalisiert werden können. Aus diesem Grund wird das im Abschnitt Fuzzy-Suche des Produkthandbuches dargestellte Beispiel:

```
SELECT subject
FROM   sample
WHERE  contains(handle, ' FUZZY FORM OF 2 "compress" ')=1
```

nie das fehlerhafte Wort *conpress* finden.

4.3.2.5 Sound-Suche

Diese Funktion soll die Suche nach Begriffen unterstützen, die den gleichen Klang wie der Suchbegriff haben, aber aufgrund unterschiedlicher Schreibweisen normalerweise nicht gemeinsam gefunden werden.

Die Durchführung diverser Tests zeigte aber, daß diese Funktion für ernsthafte Anwendungen kaum geeignet ist, da sie, ähnlich wie die Fuzzy-Suche, eine Zeichenanalyse unter Beachtung von Transpositionen und Substitutionen vornimmt. Sprachspezifische Betonungen werden dagegen kaum berücksichtigt.

Das nachfolgende Beispiel zeigt, welche Begriffe aufgrund der Sound-Suche des Terms auf der linken Seite gefunden wurden.

cause $\Longrightarrow$ cause, curse, coarse, course, case, ...
bore $\Longrightarrow$ bore, boar, bare, bar, ...
Heer $\Longrightarrow$ Heer, Haar, Heuer, ...

Die Sound-Suche wird normalerweise nur auf dem dualen oder linguistischen Index ausgeführt. Die Anwendung der Suchart auf dem präzisen oder Ngram Index sollte eine Warnung zur Folge haben, was innerhalb des Kommandozeilenprozessors auch korrekt erfolgte. Innerhalb der graphischen Benutzerumgebung des Command Centers konnte aber beobachtet werden, daß die Ausführung auf dem Ngram Index eine nichtbehandelte Exception auslöste, die das gesamte Control-Center terminieren ließ.

4.3.2.6 Proximity-Suche

Werden die Begriffe eines Suchausdrucks mittels der logischen Operatoren miteinander verknüpft, so kann jeder Begriff bei der Suche an einer beliebigen Position des Dokumentes auftreten. Wird jedoch eine gewisse Nähe der Begriffe zueinander gefordert, so daß ein semantischer Zusammenhang erhalten bleibt, kann dies über die Abstandsfunktionen erreicht werden. Die Abstandssuche des TextExtenders kann sich entweder auf einen Satz (*IN SAME SENTENCE AS*) oder einen Paragraphen (*IN SAME PARAGRAPH AS*) des Dokumentes beziehen, was aus semantischer Sicht auch plausibel erscheint.

Bei der Anwendung der Satz-Abstandssuche gibt es jedoch Probleme, wenn der Satz eine Punktierung aufgrund von Abkürzungen oder Numerierungen enthält. Bei der Indexerzeugung werden an diesen Stellen Satzgrenzen erkannt, so daß die Satz-Abstandssuche auf verkürzte Sätze angewendet wird.

4.3.2.7 Freitextsuche

Gerade für Gelegenheitsanwender eines Information-Retrieval-Systems ist es oft nicht einfach, die Konstrukte der Retrieval-Sprache so anzuwenden, daß eine effektive Suche erfolgt. Einfacher wäre es, eine natürlich-sprachliche Anfrage zu formulieren, die durch das Retrieval-System semantisch ausgewertet wird. Die semantische Erkennung von Satz-Zusammenhängen ist aber keine triviale Angelegenheit. Beim TextExtender wurde daher versucht, ein einfacheres Konzept der Freitextsuche umzusetzen. Zum einen spielt die Reihenfolge der Begriffe hier keine Rolle und zum anderen wird die sogenannte *lexikalische Nähe* unterstützt. Sie repräsentiert Wortpaare, die sowohl im Suchausdruck als auch im Dokument mit einer minimalen Häufigkeit (dies soll die Suche nach Stoppworten verhindern) und einem minimalen Abstand auftreten.

Ausgehend von der Funktionsbeschreibung der Freitextsuche sollten alle Begriffe des Suchterms mit einer minimalen Häufigkeit auch im Dokument vorkommen müssen, um in das Selektionsergebnis aufgenommen zu werden. Die Anwendung auf deutsche Dokumente zeigt aber ein anderes Verhalten, da hier das Retrieval-Ergebnis durch die Disjunktion der Dokumente, die Begriffe mit minimaler Häufigkeit enthielten, erzielt wurde.

4.3.2.8 Suche nach Synonymen

Die Suche nach Synonymformen stellt eine gute Möglichkeit dar, die Suche auf semantisch gleiche Begriffe auszudehnen. So werden bei der Synonymsuche nach dem Begriff *compute* auch die Begriffe *calculate, number, count, reckon, estimate und enumerate* in das Retrieval mit einbezogen. Laut der Produktbeschreibung des TextExtenders ist die Synonymsuche jedoch nicht in Kombination mit dem *NOT* Operator zulässig. Die nachfolgende Anfrage

```
SELECT subject
FROM   sample
WHERE contains(handle, ' NOT SYNONYM FORM OF "compute" ')=1
```

lieferte aber korrekterweise alle Dokumente, die weder den Suchbegriff noch dessen Synonymformen enthielten. Der Versuch, die Synonymsuche auf deutschsprachigen Dokumenten auszuführen, brachte leider keinen Erfolg. Anscheinend fehlte hier das deutschsprachige Synonymwörterbuch im Lieferumfang.

4.3.2.9 Thesaurus Sucherweiterung

Obwohl die Unterstützung der Synonymsuche bereits eine sehr gute Form der semantischen Anreicherung des Retrievals darstellt, werden in vielen Anwendungsdomänen erweiterte Konzepte benötigt. Der TextExtender bietet hier, mittels des Thesaurus Konzeptes die Möglichkeit, die Suche auf spezifische Anwendungsfälle auszudehnen. Hierzu ist es zunächst notwendig, die Terme der Anwendungsdomäne hierarchisch anzuordnen und zwischen den Termen Relationsbeziehungen zu bilden. Auf diese Weise können neben Synonymen auch Ober- und Unterbegriffe sowie assoziierte Terme definiert werden. Aus der SGML-Beschreibung der Terme und Relationen des Thesaurus wird ein internes Modell gebildet, das die Grundlage der Sucherweiterung innerhalb einer Anfrage darstellt. Abbildung 4.2 zeigt einen kleinen Ausschnitt aus der Hierarchie der Domäne *database management system*. Die Terme *relational database management system*, *object-oriented database management system* und *object-relational database management system* repräsentieren jeweils Unterbegriffe (NT) von *database management system*. Die Abkürzung *DBMS* ist über eine Synonymbeziehung (SYN) mit dem Begriff verbunden. *document management system* stellt einen assoziierten Term (RT) zu *database management system* dar.

Ein Retrieval des Begriffes *database management system*, welches auch das Synonym *DBMS* mit in die Suche einbeziehen soll, sieht dann folgendermaßen aus:

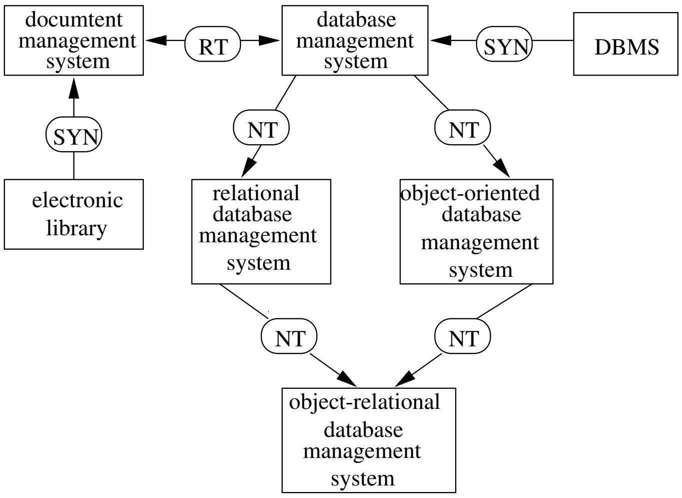

Abbildung 4.2: Beispiel einer Thesaurus-Hierarchie

```
SELECT subject
FROM   sample
WHERE  contains(handle, ' THESAURUS "thesau_db" EXPAND "SYN"
                 TERM OF "database management system" ')=1
```

4.3.3 Zusammenfassung der Funktionalität des TextExtenders

Zum Abschluß der Beschreibung des TextExtenders werden in Tabelle 4.1 die Indizes und die auf ihnen möglichen Funktionen gegenübergestellt. Ein Plus-Zeichen bedeutet dabei, daß die Funktion auf dem Index ausgeführt werden kann. Ein Minus-Zeichen signalisiert, daß die Funktion für diesen Index nicht zur Verfügung steht. Das $\oplus$-Symbol deutet auf eine eingeschränkte Anwendbarkeit der Funktion hin. Hier muß dann entweder bei der Indexerzeugung eine zusätzliche Option angegeben werden oder die Funktionalität wird nur für ausgewählte Sprachen unterstützt.

Als Fazit der Untersuchung kann gesagt werden, daß der TextExtender von der Volltext-Retrieval-Funktionalität her recht leistungsfähig ist, wenn man beachtet, daß es sich hierbei nur um einen Aufsatz auf ein objektrelationales Datenbanksystem und nicht um ein reines Volltextdatenbanksystem handelt.

Trotz des positiven Gesamteindrucks müßten einige Funktionen (Sound-Suche, Fuzzy-Suche) in einer neuen Version mit einem verbesserten Algorithmus aufwarten, um sinnvoll eingesetzt werden zu können. Bei anderen Funktionen bedarf es noch einiger Nachbearbeitung (z.B. bei der Stammformsuche), um die fehlerfreie

Funktionalität zu gewährleisten. Sollte der Hauptaspekt des TextExtenders weiterhin auf reinem Volltext-Retrieval bestehen, wäre die Erweiterung des Systems auf das Fuzzy-Retrieval-Modell sinnvoll, um durch die Retrieval-Funktion nicht nur die Retrieval-Gewichte 0 und 1 zu erhalten, sondern Werte im Intervall [0,1]. Auf dieser Basis könnten dann auch verschiedene Ranking-Algorithmen bereitgestellt werden. Günstig erscheint weiterhin die Möglichkeit mehrere Indizes auf einem Attribut zuzulassen, um die gesamte Retrieval-Funktionalität nutzen zu können. Die Unterscheidung der erzeugten Indizes dürfte sich angesichts der Nutzung von Handle-Attributen nicht schwierig gestalten. Aus der Anwendersicht, war es in vielen Situationen schwer erkennbar, welche Begriffe des Dokumentes zur Auswahl geführt haben. Eine Selektionsmöglichkeit (Highlight-Funktion) innerhalb der interaktiven Anfragesprache würde hier abhilfe schaffen.

	Linguistisch	**Präzise**	**Dual**	**Ngram**
exakte Suche	-	+	+	⊕
Stammformsuche	+	-	+	-
Fuzzy-Suche	-	-	-	+
Freitextsuche	+	+	+	-
Suche nach Synonymen	+	-	+	-
Sucherweiterung durch einen Thesaurus	+	+	+	-
Sound-Suche	+	-	+	-
Abstandssuche im gleichen Satz	+	+	+	+
Abstandssuche im gleichen Abschnitt	+	+	+	+
Bound[a]	-	-	-	+
Feature-Suche[b]	⊕	-	⊕	-

Tabelle 4.1: Überblick über Index und Suchfunktionalität des TextExtenders

[a]Die Funktionaliät von *Bound* wurde nicht untersucht, da *Bound* nur auf koreanischen Schriftzeichen anwendbar ist.

[b]Die *Feature-Suche* wurde ebenfalls nicht betrachtet, da sie nur auf Dokumenten anwendbar ist, die mit Hilfe des amerikanischen Wörterbuches indiziert wurden.

4.4 Informix Universal Server

Das seit der Version 9.1 unter dem Namen Universal Server bekannte Datenbanksystem der Firma Informix Software Inc. ist eines der führenden objektrelationalen DBMS (ORDBMS). Im Vergleich zum DB2 ORDBMS unterstützt Informix weit mehr objekt-orientierte Konzepte. Dazu zählen unter anderem

- Vererbung — Einfachvererbung in Form einer Typ- und Tabellenhierarchie
- Typkonstruktoren für Collection- und Komponenten-Datentypen
- Kapselung
- Erzeugung neuer Basisdatentypen
- funktionale Skalierbarkeit durch DataBlade-Technologie

Die im letzten Punkt genannte DataBlade-Technologie ist gekennzeichnet durch die Implementierung benutzerdefinierter Datentypen und Funktionen, wodurch Systemerweiterungen für spezifische Anwendungen möglich werden (siehe Abbildung 4.3). Mit Hilfe der DataBlade-Programmierschnittstelle (API) können DataBlade-Module auch durch Drittanbieter erstellt und anschließend durch Informix getestet und zertifiziert werden. Diese Vorgehensweise bietet dem Anwender zum einen die Flexibilität das System bei Bedarf erweitern zu können und zum anderen die Sicherheit, daß das zertifizierte DataBlade mit dem Datenbanksystem zusammenarbeitet.

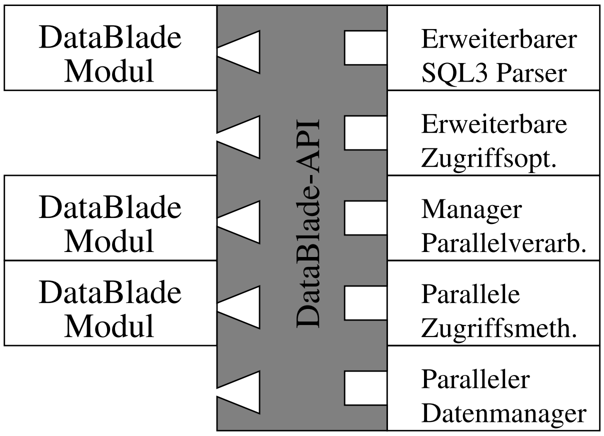

Abbildung 4.3: Schema der Datablade-Technologie

Im folgenden Abschnitt wird die Funktionalität eines dieser zertifizierten DataBlades für das Anwendungsgebiet der Volltextsuche näher untersucht. Die Auswahl dieses DataBlades erfolgte nicht nach qualitativen Gesichtspunkten, sondern aufgrund der Verfügbarkeit am Lehrstuhl für Datenbank- und Informationssysteme.

4.5 Excalibur DataBlade

Das untersuchte Excalibur Text Search DataBlade-Modul in der Version 1.10.UC2 [Inf97] der Firma Excalibur Technologies Cop. stellt Funktionen und Datentypen für das Volltext-Retrieval zur Verfügung. Neben diesem Modul, ist für die Firma Excalibur ein weiteres DataBlade für das Image-Retrieval zertifiziert. Das Excalibur-Modul besteht aus einem Textsuch-Server in Form einer Bibliothek, die einen speziellen Teil des Informix Universal Servers darstellt. Beim Aufruf einer Funktion des Volltextsystems wird durch das dynamische Binden der Excalibur-Systembibliothek das Retrieval-Modul gestartet. Durch die DataBlade-Philosophie können die Retrieval-Funktionen sowohl in Kombination mit der eingebauten DBMS-Funktionalität als auch mit anderen DataBlades verwendet werden.

Das Excalibur Text-Retrieval-Modul setzt sich aus den vier generellen Komponenten *etx-Zugriffsmethode, etx_contains()* Operator, *Filterfunktionen* und *Systemroutinen* zusammen. In den beiden nachfolgenden Abschnitten wird jedoch nur auf die Zugriffsmethode und den *etx_contains* Operator näher eingegangen.

Wie bereits beim DB2 TextExtender wurden auch für das Excalibur DataBlade die Funktionen durch Testanfragen verifiziert. Aufgetretene Differenzen zwischen der Funktionsbeschreibung und ihrem Verhalten werden in den jeweiligen Unterabschnitten dargestellt.

4.5.1 Indizierung

Für Anfragen auf unstrukturierten Dokumenten mit Hilfe des Excalibur DataBlades ist zunächst die Indizierung der Daten unter Verwendung der *etx* Zugriffsmethode erforderlich. Mit ihrer Hilfe wird ein sogenannter *etx-Index* erzeugt, der die spezifische Suche erlaubt. Innerhalb der Zugriffsmethode können verschiedene Parameter festgelegt werden, die die Charakteristik des Index und damit die auf ihm möglichen Suchfunktionalitäten festlegen. Die Erstellung des Index selbst erfolgt durch das DDL[13] Kommando *CREATE INDEX*, das in seiner Syntax um die Angabe einer Operatorklasse (siehe Tabelle 4.2) nach dem Attributnamen und einer *USING* Klausel erweitert wurde. Durch das Schlüsselwort *USING* wird die Angabe einer Zugriffsmethode eingeleitet. Die Zugriffsmethode kann entweder ein B-Baum, ein R-Baum oder eine Zugriffsmethode, die durch ein DataBlade-Modul definiert wurde, sein. Im Falle der Erzeugung eines Index für das Volltext-Retrieval mit dem Excalibur DataBlade ist hier die ext-Zugriffsmethode anzugeben. Die spezielle Operatorklasse eines durch Excalibur unterstützten Datentyps repräsentiert Funktionen für die Indexerzeugung und Anfrageoptimierung.

Durch das Beispiel

```
CREATE INDEX contentidx
ON          tabelle (content etx_varc_ops)
USING       etx (WORD_SUPPORT = 'EXACT') IN sblobspace
```

[13]DDL- data definition language

Datentyp	Operatorklasse
BLOB	etx_blob_ops
CLOB	etx_clob_ops
CHAR	etx_char_ops
VARCHAR	etx_varc_ops
LVARCHAR	etx_lvarc_ops
IfxDocDesc	etx_doc_ops

Tabelle 4.2: Unterstützte Datentypen und assoziierte Operatorklassen

wird ein Index auf dem Attribut *content* erzeugt. Die Domäne des Attributs ist vom Datentyp *varchar*, wodurch die Verwendung der Operatorklasse *etx_varc_ops* impliziert wird. In der *USING*-Klausel wurde die etx Zugriffsmethode mit dem Parameter *WORD_SUPPORT* spezifiziert.

Es wurde bereits erwähnt, daß innerhalb der Zugriffsmethode durch die Angabe von Parametern die Charakteristik des Index festgelegt wird. Im folgenden werden die Indexparameter sowie ihre möglichen Belegungswerte beschrieben.

WORD_SUPPORT : Durch diesen Parameter wird die mögliche Suchart festgelegt.

EXACT: Es wird nach der exakten Form des angegebenen Suchstrings im Anfrageausdruck gesucht. Dieser Wert ist die Standardbelegung, falls der Parameter im Index nicht spezifiziert wurde.

PATTERN: Durch den Index wird sowohl eine exakte als auch eine Fuzzy-Suche unterstützt.

PHRASE_SUPPORT : Je nach Belegungswert dieses Parameters wird eine Phrasen- und Abstandssuche ermöglicht.

NONE: Dies ist die Standardbelegung, falls der Parameter bei der Indexerzeugung nicht angegeben wurde. Es werden dann nur die Sucharten Stichwortsuche und Boolesche Suche unterstützt.

MEDIUM, MAXIMUM: Sie geben die Genauigkeit an, mit der bei der Phrasen und Abstandssuche gearbeitet werden soll. MEDIUM operiert mit einer schlechteren Genauigkeit, wobei die Wahrscheinlichkeit eines falschen Ergebnisses mit der zunehmenden Länge einer Phrase sinkt. Der Nachteil des schlechteren Suchergebnisses von MEDIUM wird jedoch durch die bessere Such-Performance aufgrund des kleineren Index kompensiert.

CHAR_SET : Die Angabe eines Zeichensatzes legt fest, welche Zeichen indexiert werden. Neben den Zeichensätzen ASCII, ISO und Overlap ISO können seit der Version 1.10.UC2 auch eigene Zeichensätze angegeben werden. Zu beachten ist, daß Zeichen des Dokumentes, die im angegebenen

Zeichensatz nicht enthalten sind, als Leerzeichen interpretiert werden, so daß unter Umständen eine Aufsplittung von Wörtern erfolgt.

Beispiel: Im ASCII-Zeichensatz ist das deutsche ß nicht enthalten, so daß *Straße* als *Stra* und *e* indexiert wird.

STOPWORD_LIST : Durch diesen Parameter wird die Indizierung, der in der spezifizierten Stoppwortliste enthaltenen Begriffe verhindert. Die angegebene Stoppwortliste muß jedoch zuvor mit Hilfe der Routine *etx_ CreateStopWlst* innerhalb eines logischen Speicherbereichs erzeugt worden sein. Die Verwendung einer Stoppwortliste ist in jedem Fall anzuraten, da dadurch die resultierende Größe des Index um einen wesentlichen Faktor reduziert wird.

INCLUDE_STOPWORDS : Dieser Parameter kann nur in Verbindung mit dem Parameter *STOPWORD_LIST* verwendet werden. Die möglichen Belegungen sind die booleschen Werte TRUE und FALSE. Durch den Wert TRUE werden die Begriffe der angegebenen Stoppwortliste zwar indiziert, jedoch werden sie nur berücksichtigt, falls im Suchausdruck der Parameter INCLUDE_STOPWORDS angegeben wurde.

Für ein Attribut einer Relation können ohne weiteres mehrere Indizes mit Hilfe der *etx*-Zugriffsmethode erzeugt werden. Überraschender Weise wird jeweils der älteste Index für die Suche verwendet. Wichtig bei der Erzeugung eines Index, ist auch zu wissen, daß prinzipiell keine sprachspezifische linguistische Verarbeitung der Begriffe existiert. Sprachabhängige Symbole werden nur durch die Wahl des Zeichensatzes bei der Indexerstellung berücksichtigt.

4.5.2 Suchfunktionalität

Bei der Beantwortung einer Information-Retrieval-Anfragen mit Hilfe der Funktionen des Excalibur DataBlades liegt das Boolean-Retrieval-Modell zugrunde. Die gesamte Suchfunktionalität des Excalibur DataBlades wird über den *etx_ contains* Operator gesteuert. Dieser Operator besitzt drei Parameter, wobei der erste und zweite Parameter obligatorisch und der dritte Parameter optional ist.

1. Parameter: Er gibt den Namen des Attributs an, auf dessen etx-Index die Suche erfolgen soll.

2. Parameter: Dieser Parameter legt den Suchausdruck fest. Er kann entweder als einfache, durch Anführungszeichen umgebene Zeichenkette oder als Reihentyp mit zwei Attributen vom Datentyp *LVARCHAR* angegeben werden. Im Falle des Reihentyps wird durch das erste Attribut das Suchkriterium festgelegt und durch das zweite Attribut können Tuning-Parameter

angegeben werden, die die Suchform beeinflussen. In den folgenden Unterabschnitten werden die einzelnen Argumente näher beleuchtet.

3. Parameter: Durch den dritten Parameter kann optional eine sogenannte *statement local variable* (SLV) angegeben werden. Sie ermöglicht es, den in der Funktionsdeklaration angegebenen Ausgabeparameter[14] in einen anderen Teil des SQL-Statements zu überführen. Der Ausgabeparameter des etx_contains Operator ist vom benannten Reihentyp *etx_ReturnType*. Die Elemente dieses Reihentyps beinhalten Ranking- und Highlight-Informationen.

4.5.2.1 Stichwortsuche

Bei der Anwendung der Stichwortsuche auf einen Index werden sämtliche Begriffe des Suchkriteriums als Disjunktion behandelt, so daß alle Tupel der Relation selektiert werden, die wenigstens einen der Begriffe des Suchterms im Index des Attributs enthalten. Die Stichwortsuche ist eine der beiden Suchformen, die in jedem Fall möglich ist, selbst wenn bei der Indexerzeugung in der Zugriffsmethode keine Argumente angegeben wurden. Wird der zweite Parameter durch eine quotierte Zeichenkette ausgedrückt, so repräsentiert die Stichwortsuche die Standard-Suchart. Im Fall des Reihentyps kann die Angabe wiederum entfallen oder sie kann im zweiten Argument des Reihentyps über das Attribut-Wert-Paar *SEARCH_TYPE=WORD* explizit bestimmt werden. Das nachfolgende Beispiel zeigt den Fall eines Reihentyps mit expliziter Angabe der Stichwortsuche.

```
SELECT subject
FROM   sample
WHERE  etx_contains(content,
       row('relationale Datenbank', 'SEARCH_TYPE=WORD'))
```

Werden innerhalb der Tuning-Parameter keine weiteren Attribute spezifiziert, so soll, gemäß des Produkthandbuches, bei der Suche ein *exact match* angewendet werden. Erstaunlicherweise wird hier nur die Reihenfolge der Buchstaben beachtet, nicht jedoch ihre Schreibweise, so daß das Wort *Datenbank* mit dem Wort *dAtENbanK* identisch ist. Da es auch keine weitere Option gibt, eine präzise Suche festzulegen, ist es zum Beispiel nicht möglich das deutsche Substantiv *Wissen* von dem Infinitiv *wissen* zu differenzieren.

4.5.2.2 Boolesche Suche

Durch die logischen Operatoren & (AND), | (OR) und ! (NOT) wird eine logische Verknüpfung der Begriffe des Suchkriteriums vorgenommen. Die Auswertungs-

[14] dieser ist nicht mit dem return type einer Funktion zu verwechseln

reihenfolge ist dabei von links nach rechts, wobei das logische *AND* stärker bindet als das logische *OR*.

Die boolesche Suche ist neben der Stichwortsuche die zweite Suchart, die auch ohne Parametrisierung der Zugriffsmethode ausgeführt werden kann. Sie wird durch die Zuweisung des Wertes *BOOLEAN_SEARCH* an den *SEARCH_TYPE*-Parameter eingeleitet.

Beispiel:

```
SELECT subject
FROM   sample
WHERE  etx_contains(content, row('relational & Datenbank',
       'SEARCH_TYPE=BOOLEAN_SEARCH'))
```

Nachteilig bei der Anwendung der booleschen Suche ist die fehlende Möglichkeit Ausdrücke zu klammern, um eine andere Auswertungsreihenfolge zu erzielen. Obwohl es vom theoretischen Standpunkt korrekt ist, daß durch die Operatoren *AND*, *OR* und *NOT* sowie der Vorrangregel für *AND* jegliche boolesche Funktionen durch die Darstellung als disjunktive Normalform ohne Klammerung beschrieben werden können, wird durch diese Darstellung weder Übersichtlichkeit noch einfache Anwendbarkeit erreicht.

4.5.2.3 Phrasensuche

Im Kontext des Excalibur DataBlades stellt eine Phrase ein Suchkriterium dar, das mehr als einen Begriff enthält. Im Kontrast zur Stichwortsuche, bei der die Begriffe des Suchkriteriums getrennt voneinander gesucht werden, bilden die Begriffe bei der Phrasensuche eine Einheit.

Durch das DataBlade-Modul werden zwei Formen der Phrasensuche unterstützt. Zum einen ist dies die exakte Phrasensuche, bei der die einzelnen Begriffe des Suchausdrucks in der selben Reihenfolge im Dokument auftreten müssen. Sie wird durch die Zuweisung des Wertes *PHRASE_EXACT* an den *SEARCH_TYPE*-Parameter eingeleitet. Die zweite Form ist die approximierte Phrasensuche, in der weder die Reihenfolge der Begriffe, noch ihre exakte Form eine maßgebliche Rolle spielt. Die Durchführung einer approximierten Phrasensuche wird durch das Attribut-Wert-Paar *SEARCH_TYPE=PHRASE_APPROX* ermöglicht. Im Unterschied zur Stichwort- und booleschen Suche, die auf allen etx-Indizes möglich sind, kann die Phrasensuche nur auf einem etx-Index operieren, der mit dem Parameter *PHRASE_SUPPORT* erzeugt wurde.

Im Excalibur Produkthandbuch wird von der Verwendung der approximierten Phrasensuche aufgrund der schlechten Performance abgeraten. Dies konnte bei den durchgeführten Tests durch die zu geringe Dokumentenanzahl und -größe nicht beobachtet werden, jedoch war bei den Testanfragen die Retrieval-Qualität teilweise recht schlecht. Für die approximierte Phrasensuche reichte es in jedem Fall aus, wenn ein Begriff der Phrase indiziert war. In dem Fall des folgenden Beispiels wurden alle Dokumente selektiert, die den Begriff *Datenbank* enthielten.

```
SELECT subject
FROM   sample
WHERE  etx_contains(content, row('xxx Datenbank yyy ',
       'SEARCH_TYPE=PHRASE_APPROX'))
```

4.5.2.4 Proximity-Suche

Wie bereits erwähnt, betrachten die Stichwort- und die Boolesche Suchform die einzelnen Terme getrennt voneinander, so daß auch der Kontext, in dem sie sich befinden, unterschiedlich sein kann. Eine Kontextabhängigkeit wird in gewisser Weise durch die exakte Phrasensuche erzielt, die aber durch die Forderung, daß sämtliche Begriffe in der angegebenen Reihenfolge auftreten müssen, im allgemeinen zu restriktiv ist. In vielen Situationen ist es so, daß zwar der semantische Zusammenhang gewünscht wird, eine exakte Formulierung der Phrase jedoch nicht angegeben werden kann. Ein Ausweg in Excalibur ist die Proximity-Suchform. Hier kann neben der Phrase ein Abstandsparameter spezifiziert werden, der die maximale Anzahl an Worten angibt, die zwischen den Begriffen des Suchkriteriums auftreten dürfen, wobei die beiden Begriffe selbst auch mitgezählt werden. In die Abstandsberechnung werden sämtliche Begriffe des Dokumentes mit einbezogen. Neben der geforderten Nähe der Begriffe zueinander, müssen sämtliche Terme im Dokument auftreten, nicht jedoch in der angegebenen Reihenfolge.

Beispiel:

```
SELECT subject
FROM   sample
WHERE  etx_contains(content, row('relationale Datenbank',
       'SEARCH_TYPE=PROX_SEARCH(5)'))
```

4.5.2.5 Fuzzy-Suche

In den bisherigen Untersuchungen wurde davon ausgegangen, daß die Terme des Suchkriteriums in ihrer exakten Form (bzw. in der durch Excalibur unterstützten exakten Form, siehe Abschnitt 4.5.2.1) im Dokument auftraten. Die volle Mächtigkeit eines Volltext-Retrieval-Systems zeigt sich jedoch erst, wenn gewisse unscharfe Suchanfragen möglich werden, die neben der exakten Suche auch Suchanfragen nach Begriffen mit orthographischen Fehlern bzw. die Suche nach

Wortteilen zulassen. Um solch eine Suche mit Hilfe des *etx_contains*-Operators ausführen zu können, muß bei der Erzeugung des etx-Index der Parameter *WORD_SUPPORT* der etx-Zugriffsmethode mit dem Wert *PATTERN* belegt worden sein. Die größere Flexibilität in den Retrieval-Anfragen wird dann durch die Verwendung der vier verschiedenen Fuzzy-Parameter erreicht.

PATTERN_TRANS : Die Ergebnismenge der Anfrage enthält Dokumente, bei denen das Suchkriterium entweder exakt im Dokument auftritt oder deren Terme genau eine Transposition (Vertauschung zweier benachbarter Zeichen) gegenüber den Termen des Suchkriteriums aufweisen.

PATTERN_SUBS : Die Ergebnismenge der Anfrage enthält Dokumente, bei denen das Suchkriterium entweder exakt im Dokument auftritt oder deren Terme genau eine Substitution (Ersetzung eines Zeichens durch ein anderes) gegenüber den Termen des Suchkriteriums aufweisen.

PATTERN_BASIC : Durch die Parametrisierung einer Anfrage mit *PATTERN_BASIC* werden neben der spezifizierten Form des Suchausdrucks auch Ober- und Untermengen des Suchmuster im Index gesucht.

PATTERN_ALL : Dieser Parameter repräsentiert die höchste Abstraktionsebene der Fuzzy-Suche. Hier wird das Suchkriterium auf Begriffe mit multiplen Transpositionen und Substitutionen, sowie auf Ober- und Untermengen des Suchmusters erweitert.

Im Handbuch zum Informix-DataBlade wurde keinerlei Aussage darüber getroffen, wie weit die Begriffe Ober- bzw. Untermenge zu ziehen sind. Durch Testanfragen stellte sich heraus, daß ein Oberbegriff maximal zwei Zeichen am Wortanfang und drei Zeichen am Wortende über den Suchbegriff hinausgehen darf, so daß die Retrieval-Anfrage des Handbuches auf Seite 1-29:

```
SELECT subject
FROM   sample
WHERE  etx_contains(content,
       row('retrieve', 'PATTERN_BASIS'))
```

zwar korrekterweise die Begriffe *retrieve* und *retrieving*, nicht jedoch den angegebenen Begriff *irretrieveable* finden wird. Desweiteren sollen durch diesen Parameter Transpositionen und Substitutionen unbeachtet gelassen werden. Dies ist jedoch nicht der Fall, da beispielsweise auch der fehlerhafte Begriff *retreive* gefunden wird.

4.5.2.6 Suche nach Synonymen

Ist der Inhalt der Dokumente, auf die sich eine Suchanfrage bezieht im vorhinein unbekannt, so ist es hilfreich, die Anfrage durch eine Synonymwortliste anzureichern, so daß neben den eigentlichen Suchtermen auch sinnverwandte Begriffe in die Suche einbezogen werden. Im Excalibur DataBlade kann die Synonymsuche über den Parameter *MATCH_SYNONYM* gesteuert werden. Wird diesem Parameter kein Wert zugewiesen, so wird die Standardsynonymliste verwendet. Sie enthält jedoch nur Synonyme der englischen Sprache. Sollen Synonyme einer anderen Sprache zur Anwendung kommen oder ein erweiterter englischer Wortschatz, so kann über die Systemroutine *etx_ CreateSynWlst* eine eigene Synonymwortliste hinzugefügt werden. Eine Excalibur-Synonymliste besteht aus mehreren Wortreihen. Das erste Wort jeder Reihe stellt den Descriptor dar; alle weiteren Worte sind sinnverwandte Begriffe des Descriptors.

Beispiel:

PENSION FERIENWOHNUNG APPARTEMENT

Damit eine Suchanfrage um die Synonyme eines Begriffe angereichert werden kann, muß dieser Begriff in der Synonymliste als Descriptor erscheinen und die Suchanfrage darf keinen Fuzzy-Parameter enthalten.

Beispiel:

```
SELECT subject
FROM   sample
WHERE  etx_contains(content,
       row('Pension', 'MATCH_SYNONYM=syn_deutsch'))
```

Leider war im gesamten Handbuch kein Hinweis darauf zu finden, daß sämtliche Begriffe der Synonymliste in Großbuchstaben zu schreiben sind, so daß es einiger Versuche bedurfte, ehe die erste Suche auch eigene Synonyme mit einbezog.

4.5.2.7 Ranking der Ergebnisse

Der Indikator für die Güte eines Dokumentes zur gestellten Anfrage ist ein Wert zwischen 0 und 100. Beim Wert Null fand die Anfrage in den entsprechenden Dokumenten keinen Treffer. Der Wert 100 spiegelt das exakte Vorkommen des Suchkriteriums im Dokument wieder. Alle Werte zwischen diesen Extremwerten deuten darauf hin, daß das Suchkriterium nur in einer approximierten Form im Dokument auftritt. Für die Berechnung der Ranking-Werte werden die folgenden zwei Regeln angewendet:

1. Exakte Übereinstimmungen werden immer geringfügig besser bewertet als Pattern-Matches.
2. Alle Pattern-Matches werden gleich bewertet.

Zu Beginn des Abschnittes 4.5.2 wurden die Argumente des etx_contains Operator erläutert. Als drittes Argument kann hier eine optionale *statement local variable* angegeben werden, durch die die Ranking-Werte in anderen Teilen des SQL-Statements verwendet werden können.

Obwohl es generell schwierig, ist beim Boolean-Retrieval-Modell eine aussagekräftige Rangliste zu bilden, wäre es vielleicht sinnvoll, auch die Vorkommenshäufigkeiten mit in die Rangliste einzubeziehen. So sind die Ranglistenwerte einer booleschen Anfrage, deren Terme durch Konjunktionen verbunden sind und die keine Fuzzy-Parameter enthält, generell 100.

4.5.2.8 Result-Highlighting

Sämtliche Begriffe eines Dokumentes, die durch das Suchkriterium einer Retrieval-Anfrage repräsentiert werden, können über die Funktion *etx_ GetHilite* ausgegeben werden. Der Rückgabetyp dieser Funktion ist ein Reihentyp mit den Feldern *vec_ offset* und *viewer_ doc*. Das Feld *vec_ offset* enthält für jeden übereinstimmenden Begriff die absolute Anfangsposition im Dokument und die Länge des Begriffes. Das Feld *viewer_ doc* beinhaltet das von Formatierungsparametern befreite Dokument.

4.5.3 Zusammenfassung der Funktionalität des DataBlades

In der Tabelle 4.3 wird noch einmal der Zusammenhang zwischen den unterstützten Sucharten und den beiden Indexparametern *WORD_ SUPPORT* und *PHRASE_ SUPPORT* verdeutlicht. Ein Plus beim *WORD_ SUPPORT* verweist darauf, daß die Suchart mit dieser Belegung des Indexparameters harmoniert. Die Angaben zu *Phrase_ Support* bestimmen die Notwendigkeit dieses Parameters bei der Indexerzeugung, falls die entsprechende Suchart zum Einsatz kommt.

Als Fazit der Untersuchung des Excalibur DataBlades sind viele Restriktionen hinsichtlich der Retrieval-Funktionalität zu konstatieren. Abgesehen vom Fehlen einer Stammformreduktion können auch exakte Anfragen nur in der in Abschnitt 4.5.2.1 genannten Weise durchgeführt werden. Desweiteren beziehen sich sämtliche Tuning-Parameter einer Retrieval-Anfrage stets auf das gesamte Suchkriterium, so daß eine spezifische Formulierung für einzelne Terme nur durch eine getrennte Anfrage realisierbar ist. Diese darf aber wiederum nicht mit weiteren *etx_ contains* Operatoren in einer WHERE-Klausel kombiniert werden. Der negative Eindruck aufgrund des begrenzten Funktionsumfangs wird noch verstärkt

Suchart	WORD_SUPPORT		Phrase-Support
	EXACT	PATTERN	
Stichwort-Suche	+	+	nicht erforderlich
Boolesche Suche	+	+	nicht erforderlich
Fuzzy-Suche	-	+	nicht erforderlich
Phrasensuche	+	+	erforderlich
Proximity-Suche	+	+	erforderlich

Tabelle 4.3: Überblick über ausgewählte Indexparameter und die Suchfunktionalität

durch Funktionen, die ein von der Spezifikation abweichendes Retrieval-Ergebnis liefern (z.B. PATTERN_BASIC).

4.6 Vergleich der Retrieval-Funktionalität

Abschließend sollen die beiden Volltexterweiterungen einem Vergleich unterzogen werden. Als Vergleichsmaßstab dienen im wesentlichen die in [Rah98] verwendeten Kriterien. Neben der Unterstützung der Kriterien durch das System soll, insofern möglich, die Anwendbarkeit beurteilt werden. Maßstab für eine Anwendbarkeit ist schwerpunktmäßig die durch die Untersuchung festgestellte Retrieval-Qualität. Einfluß haben aber auch Faktoren wie beispielsweise die Handhabung.

Die Einordnung der Systeme hinsichtlich der Unterstützung der Kriterien erfolgt nach folgenden Gesichtspunkten.

$\oplus$ Das Kriterium wird durch das System unterstützt.
$\otimes$ Das Kriterium kann nur simuliert werden.
$\ominus$ Das Kriterium wird nicht durch das System unterstützt.

Die Beurteilung der Anwendbarkeit wird aufgrund folgender Klassifikation vorgenommen

$+$ Sehr gute Anwendbarkeit — Die Retrieval-Qualität entsprach den Erwartungen bzw. es wurden keine Einschränkungen festgestellt.
$\pm$ Gute Anwendbarkeit — Die Retrieval-Qualität war überwiegend gut bzw. es wurden nur geringfügige Einschränkungen festgestellt.
$\mp$ Eingeschränkte Anwendbarkeit — Die Retrieval-Qualität war nur ausreichend bzw. es wurden größere Einschränkungen festgestellt.
$-$ Unzureichende Anwendbarkeit — Die Anwendung der Funktion wird aufgrund unzureichender Retrieval-Ergebnisse oder anderer Einschränkungen nicht empfohlen.

Retrieval-Modell: Sowohl durch den TextExtender als auch durch Excalibur wird lediglich das Boolean-Retrieval-Modell unterstützt, so daß für beide Systeme auch die in Abschnitt 4.1 aufgeführten Nachteile gelten.

Indizierung: Beide Systeme arbeiten beim Volltext-Retrieval nicht auf den Datenbeständen selbst, sondern auf einem speziellen Index. Klarer Vorteil des Excalibur-Systems ist, daß der Index nicht vom Datenbanksystem getrennt ist und nach Änderungsoperationen automatisch angepaßt wird. Der TextExtender wiederum wartet für die Indizierung mit umfangreicher Sprachunterstützung auf, die in Excalibur nur durch einen entsprechenden Zeichensatz simuliert werden kann.

Excalibur: *Unterstützung* $\oplus$
TextExtender: *Unterstützung* $\oplus$

Stammformreduktion: Die Stammformreduktion ist einer der Kernpunkte für eine linguistische Suche. Erst dadurch wird es möglich, die Suche auf alle Begriffsformen (Zeitformen, Deklinationen) zu erweitern. Der TextExtender führt eine Stammformreduktion bei der Erstellung des linguistischen und des dualen Index durch, sowie auf den Termen von Anfragen, die auf diesen Indizes operieren. Durch die in Abschnitt 4.3.2.3 festgestellten fehlerhaften Retrieval-Ergebnisse wird die Anwendbarkeit der Stammformsuche des TextExtenders allerdings abgewertet. Durch Excalibur wird keine Stammformreduktion unterstützt. Das Fehlen der Stammformreduktion stellt eine große Schwäche des Excalibur Systems dar, da die linguistische Suche einen der Hauptaspekte des Text-Retrievals repräsentiert, um Unschärfen in Anfragen zuzulassen. In Excalibur kann nur eine sehr eingeschränkte Simulierung der Stammformsuche über den Fuzzy-Parameter *PATTERN_BASIC* erfolgen.

Excalibur: *Unterstützung* $\ominus$
TextExtender: *Unterstützung* $\oplus$, *Anwendbarkeit* $\pm$

Natürlich-sprachliche Anfragen: Eine natürlich-sprachliche Anfrage kann zwar in beiden Systemen formuliert, aber nicht auf natürlich-sprachliche Weise verarbeitet werden. Im TextExtender kann eine solche Anfrage unter Verwendung der Freitextsuche (Abschnitt 4.3.2.7) auf Begriffshäufigkeiten und -Abstände untersucht und verarbeitet werden. Die semantischen Zusammenhänge der Begriffe werden aber nicht erkannt, so daß das Retrieval-Ergebnis u.U. fehlerhaft ist. Im Excalibur-System kann eine natürlich-sprachliche Anfrage nur als Phrasen- oder Stichwortsuche ausgewertet werden, was ebenfalls zu falschen Retrieval-Ergebnisse führen kann.

Excalibur: *Unterstützung* $\otimes$, *Anwendbarkeit* $\mp$
TextExtender: *Unterstützung* $\otimes$, *Anwendbarkeit* $\mp$

Thesaurus: Der Vergleich des Kriteriums Thesaurus fällt klar zu Gunsten des TextExtenders aus. Neben der Erweiterung einer Anfrage auf Synonymformen der enthaltenen Terme kann zusätzlich für spezielle Anwendungsgebiete ein Thesaurus erzeugt werden (Abschnitt 4.3.2.9), der die Suche um zusätzliche Relationsbeziehungen erweitert. In Excalibur kann eine Anfrage hingegen nur um Synonymformen erweitert werden.

Excalibur: *Unterstützung* $\oplus$, *Anwendbarkeit* $\pm$
TextExtender: *Unterstützung* $\oplus$, *Anwendbarkeit* $+$

Boolesche Suche: Die boolesche Suche wird durch beide Systeme unterstützt und die Anfrageergebnisse entsprachen den zu erwartenden Resultaten. Das Excalibur-System wird in der Bewertung der Anwendbarkeit jedoch etwas herabgesetzt, da aufgrund der fehlenden Schachtelung die Frageformulierung umfangreicher Anfragen kompliziert ist.

Excalibur: *Unterstützung* $\oplus$, *Anwendbarkeit* $\pm$
TextExtender: *Unterstützung* $\oplus$, *Anwendbarkeit* $+$

Proximity-Suche: Funktionen zur Abstandssuche werden durch beide Systeme bereitgestellt. Der semantische Zusammenhang tritt durch die Funktionen des TextExtenders (Abschnitt 4.3.2.6) deutlicher hervor, jedoch führt die Erkennung fehlerhafter Satzgrenzen zur Abwertung, da hierdurch das Retrieval-Ergebnis verfälscht wird. Im Fall von Excalibur kann durch Angabe eines zu großen Abstandswertes der gemeinsame Kontext der Terme nicht gewährt sein, so daß die Abstandssuche im schlechtesten Fall zur Stichwortsuche ausartet.

Excalibur: *Unterstützung* $\oplus$, *Anwendbarkeit* $\pm$
TextExtender: *Unterstützung* $\oplus$, *Anwendbarkeit* $\pm$

Sound-Suche: Diese Suchform steht nur im TextExtender zur Verfügung. Aufgrund der in Abschnitt 4.3.2.5 aufgeführten Schwächen im Retrieval wird die Anwendbarkeit dieser Funktion als gering bewertet.

Excalibur: *Unterstützung* $\ominus$
TextExtender: *Unterstützung* $\oplus$, *Anwendbarkeit* $\mp$

Fuzzy-Suche: Diese Suchform steht in beiden Systemen zur Verfügung. Aufgrund des Fehlens einer Stammform-Suche wartet das Excalibur-System hier mit einer umfangreicheren Funktionalität auf, die jedoch an einigen Stellen nicht völlig fehlerfrei arbeitet. Die Fuzzy-Suche des TextExtenders arbeitet nur mit dem Ngram-Index. Des weiteren gilt die Einschränkung, daß mindestens die ersten drei Buchstaben des Wortes mit dem Suchterms übereinstimmen müssen.

Excalibur: *Unterstützung* $\oplus$, *Anwendbarkeit* $\pm$
TextExtender: *Unterstützung* $\oplus$, *Anwendbarkeit* $\mp$

Phrasensuche: Eine exakte Phrasensuche ist in beiden Systemen möglich. Die Bewertung der Excalibur-Funktion wird aber geringfügig abgewertet, da trotz exakter Suche keine Differenzierung zwischen Groß- und Kleinbuchstaben erfolgt. Die Suche nach einer Phrase, deren Terme eine andere Reihenfolgen besitzen dürfen, kann im TextExtender relativ gut über die Proximity-Suche simuliert werden. Im Excalibur existiert hierfür die Funktion der approximierten Phrasensuche, die jedoch die Phrasensuche zu großzügig auffaßt, wodurch ein relativ schlechtes Retrieval-Ergebnis die Folge ist.

Excalibur: *Unterstützung* $\oplus$, *Anwendbarkeit* $\pm$ / $-$
TextExtender: *Unterstützung* $\oplus$ / $\otimes$, *Anwendbarkeit* $+$ / $\pm$

Zeichenmaskierung: Der TextExtender bietet zwei Zeichen zur Maskierung an. Durch % wird eine beliebige Anzahl Zeichen und durch _ genau ein Zeichen an der entsprechenden Stelle maskiert. Die Zeichenmaskierung eignet sich jedoch nur für den präzisen und dualen Index, da im Fall des linguistischen Index ein maskiertes Zeichen nicht auf die Stammform reduziert werden kann. Durch Excalibur kann die Zeichenmaskierung nur mit Hilfe des Fuzzy-Parameters zur Substitution simuliert werden. Auf diese Weise kann aber maximal ein Zeichen eines Terms maskiert werden.

Excalibur: *Unterstützung* $\otimes$, *Anwendbarkeit* $\mp$
TextExtender: *Unterstützung* $\oplus$, *Anwendbarkeit* $\pm$

Ranking: Beide Systeme stellen genau eine Ranking-Funktion (Abschnitt 4.3.2 und 4.5.2.7) zur Verfügung, jedoch ist keine der beiden Funktionen ideal. Die Ursache ist hier im verwendeten Retrieval-Modell zu sehen, das nur eine nachträgliche Bewertung der Ergebnisse erlaubt. Die exklusive Bewertung der Vorkommenshäufigkeiten des TextExtender hat z.B. zur Folge, daß eine Stammform eines Terms, die mehrfach in einem Dokument auftritt, in der Rangfolge besser bewertet wird als ein exakter Treffer, der genau einmal im Dokument enthalten ist (Voraussetzung ist, daß beide Dokumente eine ähnliche Größe aufweisen).

Excalibur: *Unterstützung* $\oplus$, *Anwendbarkeit* $\mp$
TextExtender: *Unterstützung* $\oplus$, *Anwendbarkeit* $\mp$

Relevance Feedback: Aufgrund des verwendeten Retrieval-Modells wird Relevance Feedback weder durch den TextExtender, noch durch Excalibur unterstützt. Im TextExtender steht lediglich eine Funktion zur Verfügung, die eine optimierte Form der Anfrageverfeinerung vornimmt. Die Verfeinerung der Anfrage erfolgt auf dem Retrieval-Ergebnis der zuvor gestellten Anfrage. Eine Anfrageverfeinerung ist auch in Excalibur über die Datenbankfunktionalität zu realisieren. Betont werden muß jedoch, daß die Verfeinerung einer Anfrage nicht dem in Abschnitt 4.1 beschriebenen Relevance-Feedback entspricht.

Excalibur: *Unterstützung* $\otimes$
TextExtender: *Unterstützung* $\otimes$

Result-Highlighting: Die größte Einschränkung des TextExtenders gegenüber Excalibur stellt das Fehlen einer Funktion dar, die Aufschluß über die Terme gibt, die zur Auswahl des entsprechenden Dokumentes geführt haben. Im Excalibur-System kann dies auf die in Abschnitt 4.5.2.8 beschriebene Weise erfolgen.

Excalibur: *Unterstützung* $\oplus$, *Anwendbarkeit* $+$
TextExtender: *Unterstützung* $\ominus$

Kombinierte Text- /Attributanfragen: Die kombinierte Anfrage über Attribute der Datenbank und Text-Retrieval ist weniger ein Merkmal der Volltext-

Erweiterungen als des Datenbanksystems. Durch beide Datenbanksysteme sind kombinierte Anfragen möglich, jedoch erlaubt Informix nicht die mehrfache Aufführung der selben nutzerdefinierten Funktion in der WHERE-Klausel.

Excalibur: *Unterstützung* $\oplus$, *Anwendbarkeit* $\pm$
TextExtender: *Unterstützung* $\oplus$, *Anwendbarkeit* $+$

Speicherung des Dokumentes: Beide Systeme ermöglichen die Speicherung der Dokumente sowohl im Datenbanksystem als auch im Dateisystem.

Excalibur: *Unterstützung* $\oplus$
TextExtender: *Unterstützung* $\oplus$

Unterstützung verschiedener Textformate: Die Filtermöglichkeit für diverse Dateiformate ist in beiden Systeme äußerst eingeschränkt.

Excalibur: *Unterstützung* $\oplus$, *Anwendbarkeit* $\mp$
TextExtender: *Unterstützung* $\oplus$, *Anwendbarkeit* $\mp$

In Tabelle 4.4 werden die Ergebnisse des zuvor durchgeführten Vergleichs zwischen den Volltext-Retrieval-Systemen des TextExtenders und des Excalibur DataBlades noch einmal zusammengefaßt. Die in der Tabelle verwendeten Symbole entsprechen den Symbolen der Kriterien für die Unterstützung und Anwendbarkeit der Funktionen.

Quintessenz der Evaluierung der Volltext-Retrieval-Funktionalität ist, daß der TextExtender von der Funktionalität her wesentlich leistungsfähiger ist, als das Excalibur DataBlade. Dies betrifft insbesondere die sprachspezifische linguistische Suche des TextExtenders, die Möglichkeit der gezielten Attributierung einzelner Terme und die Sucherweiterungen mit Hilfe eines Thesaurus, der über die Möglichkeiten einer Synonymwortliste hinausgeht. Sollte beispielsweise für die Konzeption einer Suchmaschine die Wahl getroffen werden, auf welches Volltext-Retrieval-System Anfragen abgebildet werden sollen, so ist aufgrund der Analyse eindeutig der TextExtender zu bevorzugen, da das Excalibur DataBlade zu viele Restriktionen hinsichtlich der Funktionalität aufweist. Betont werden muß aber, daß sich die Bewertung allein auf die Volltexterweiterungen beziehen und nicht auf das jeweilige Datenmodell. Sollte eine Entscheidung hinsichtlich des Datenmodells gefällt werden, würde die Entscheidung zu Gunsten von Informix ausfallen.

	TextExtender		**Excalibur**	
	Unterstütz.[a]	*Anwendb.*[b]	*Unterstütz.*	*Anwendb.*
Retrieval-Modell				
Indizierung	$\oplus$		$\oplus$	
Stammformreduktion	$\oplus$	$\pm$	$\ominus$	
nat.-sprachl. Anfragen	$\otimes$	$\mp$	$\otimes$	$\mp$
Thesaurus	$\oplus$	$+$	$\oplus$	$\pm$
Boolesche Suche	$\oplus$	$+$	$\oplus$	$\pm$
Proximity-Suche	$\oplus$	$\pm$	$\oplus$	$\pm$
Sound-Suche	$\oplus$	$\mp$	$\ominus$	
Fuzzy-Suche	$\oplus$	$\pm$	$\oplus$	$\mp$
Phrasensuche	$\oplus$ / $\otimes$	$+$ / $\pm$	$\oplus$	$\pm$ / $-$
Zeichenmaskierung	$\oplus$	$\pm$	$\otimes$	$\mp$
Ranking	$\oplus$	$\mp$	$\oplus$	$\mp$
Relev. Feedback	$\otimes$		$\otimes$	
Result-Highlighting	$\ominus$		$\oplus$	$+$
Text-/Attributanfr.	$\oplus$	$+$	$\oplus$	$\pm$
Speicherung	$\oplus$		$\oplus$	
Textformate	$\oplus$	$\mp$	$\oplus$	$\mp$

[a] Folgende Unterstützungskriterien wurden angewendet:

$\oplus$	Das Kriterium wird durch das System unterstützt.
$\otimes$	Das Kriterium kann nur simuliert werden.
$\ominus$	Das Kriterium wird nicht durch das System unterstützt.

[b] Die Beurteilung der Anwendbarkeit wird aufgrund folgender Klassifikation vorgenommen

$+$	Sehr gute Anwendbarkeit
$\pm$	Gute Anwendbarkeit
$\mp$	Eingeschränkte Anwendbarkeit
$-$	Unzureichende Anwendbarkeit

Tabelle 4.4: Vergleich der Retrieval-Funktionaltät von DB2 und Informix

Kapitel 5

Prototypische Datentyperweiterung

Angesichts der Charakteristika semistrukturierter Daten (vgl. Kapitel 2) wie beispielsweise der *Flexibilität*, der *Implizität* oder der *Unregelmäßigkeit des Schemas*, gestaltet sich eine Umsetzung dieser Strukturen in ein objektrelationales Datenbankschema schwierig. Mittels der Skalierbarkeit ORDBS könnten dennoch Anfragen auf diesen impliziten Strukturen gewährleistet werden, indem die Volltexterweiterungen objektrelationaler DBMS diese Struktur- und Inhaltsanfragen durchführen. Für diese Diplomarbeit bedeutet dies, daß die Volltexterweiterungen in der Lage sein sollten, eine *Differenzierung* zwischen den *XML-Strukturelementen* und ihren *Inhalten* vorzunehmen. Die im vorherigen Kapitel durchgeführte Untersuchung der Volltexterweiterungen objektrelationaler Datenbanksysteme anhand der Systemerweiterungen von DB2 und Informix hat bisher nur die generellen Möglichkeiten dieser Systeme in Bezug auf das Volltext-Retrieval ins Auge gefaßt. Die Frage, inwieweit sie sich auch für das Retrieval semistrukturierter Dokumente[1] eignen, wurde bewußt außer Acht gelassen. Im weiteren Verlauf dieses Kapitels wird diese Frage geklärt. Dazu werden zunächst Kriterien aufgestellt, die für das Volltext-Retrieval auf semistrukturierten Dokumenten von Bedeutung sind (Abschnitt 5.1). Daran anschließend wird die Verwendbarkeit der Retrieval-Module des DB2 DBMS in Abschnitt 5.2 und des Informix DBMS in Abschnitt 5.3 hinsichtlich der Anforderungskriterien analysiert. Darauf aufbauend wird für beide Systeme eine Konzeption erarbeitet, die die nicht erfüllten Anforderungskriterien in die Volltexterweiterungen integriert. Für den TextExtender des DB2 Datenbanksystems wird neben der Konzeption eine Implementierung vorgestellt, die einen Teil der Konzeption umsetzt.

[1]Wenn im weiteren Verlauf dieses Kapitels von semistrukturierten Dokumenten die Rede ist, so sind stets XML-Dokumente gemeint.

5.1 Anforderungskriterien

Zur Bewertung der untersuchten Retrieval-Systeme hinsichtlich der Anwendbarkeit für das Retrieval auf XML-Dokumenten werden nachfolgend Anforderungskriterien festgelegt. Ausschlag für die Formulierung der Kriterien gaben die spezifischen XML-Konstrukte und deren logische Struktur. Datenbankspezifische Kriterien wie Join-, Mengen- oder Aggregatoperationen wurden hier nicht betrachtet. Für die aufgestellten Anforderungskriterien wird kein Anspruch auf Vollständigkeit erhoben, vielmehr bilden sie den Maßstab für die Konzeption der XML-Retrieval-Module.

Bei der Beschreibung der Anforderungskriterien wird zwischen Anfrage- und Änderungsoperationen unterschieden. Änderungsoperationen haben nur indirekt Einfluß auf das Retrieval semistrukturierter Dokumente. Kriterien für diese Operationen repräsentieren Integritätsbedingungen, die einen konsistenten Datenbankzustand gewährleisten. Kriterien für Anfrageoperationen sind u.a. Anfragemöglichkeiten an Attributwerte und Elementinhalte, an die hierarchische und sequentielle Struktur sowie die Interpretation von Inhalten, Elementen und Attributwerten.

Kriterien für Änderungsoperationen

1. *Update mit DTD:* Änderungsoperationen auf Strukturelementen von Dokumenten, die Instanzen einer DTD darstellen, dürfen die Konformität dieses Dokumentes hinsichtlich seiner DTD nicht verletzen.
2. *Update ohne DTD:* Änderungsoperationen auf Strukturelementen von Dokumenten ohne DTD dürfen die Wohlgeformtheit des Dokumentes nicht verletzen.
3. *Indexaktualisierung nach Update:* Änderungsoperationen auf den XML-Dokumenten müssen einen Index-Update nach sich ziehen.
4. *Insert mit DTD:* Einfügeoperationen müssen Dokumente ohne DTD auf Wohlgeformtheit prüfen.
5. *Insert ohne DTD:* Einfügeoperationen von Dokumenten mit DTD erfordern eine Validierung des Dokumentes hinsichtlich seiner DTD.
6. *Indexaktualisierung nach Insert:* Einfügeoperationen auf den Attributen der XML-Dokumente müssen einen Index-Update nach sich ziehen.
7. *Indexaktualisierung nach Delete:* Löschoperationen auf den Attributen der XML-Dokumente müssen einen Index-Update nach sich ziehen.

Kriterien für Anfrageoperationen

1. *Trennung Struktur/Inhalt:* Die Anfragesprache sollte eine Trennung zwischen Strukturelementen und ihren Inhalten gewährleisten (Bsp.: Gibt es

ein Strukturelement "Kapitel" im Dokument? bzw. Gibt es Elemente die den Begriff "Kapitel" enthalten?).

2. *Attributanfragen:* Durch die Anfrageoperationen sollte ein Zugriff über Attribute von Elementen realisierbar sein (Bsp.: Finde Elemente die das Attribut "Sprache" mit dem Wert "deutsch" enthalten).
3. *Bereichsauswahl:* Anfragen sollten eine gezielte Auswahl eines logischen Strukturbereichs ermöglichen (z.B. durch Pfadausdrücke).
4. *Abstandsanfragen:* Durch die Anfrageoperationen sollten Abstandsanfragen realisierbar sein. Dabei kann sich die Distanz auf Terme beziehen, die gemeinsam im Inhalt eines Elementes enthalten sein müssen oder auf den Abstand von Elementen untereinander.
5. *Inklusionsbeziehungen:* Durch die Anfrageoperationen sollten Anfragen zu Inklusionsbeziehungen realisierbar sein (Bsp.: Finde Elemente, die andere Elemente enthalten oder die Begriffe in einer definierten Häufigkeit enthalten).
6. *Typisierung:* Neben der Gleichheit auf Zeichenketten sollten durch die Anfrageoperationen auch Gleichheit und größer/kleiner-Beziehungen für beliebige Typen bei Attribut- und Inhaltsanfragen berücksichtigt werden. Eine Typangabe im Anfrageausdruck bewirkt die Transformation der Werte in den Zieltyp.

   Beisp.: *Gibt es Hotels mit Zimmerpreisen unter 100,00 DM?*
   Der Preis ist hier Inhalt des Elementes Zimmerpreis. In der Anfrage muß dann für den Interpretationstyp "real" angegeben werden.
7. *Indizierung:* Der Indizierungsprozeß sollte sämtliche Strukturelemente bei der Indizierung berücksichtigen, auch solche, die bei Verwendung einer Stoppwortliste eliminiert werden würden.

5.2 Konzeption / Implementierung für DB2 TextExtender

Vor der konzeptionellen Erweiterung des TextExtenders auf das Retrieval semistrukturierter Dokumente erfolgt in Abschnitt 5.2.1 zunächst eine Validierung des TextExtenders hinsichtlich der Anforderungskriterien. Abschnitt 5.2.2 zeigt dann einige Probleme, die bei der Abbildung von XML-Strukturen auf Datenbankstrukturen entstehen können. Die anschließende Konzeption setzt sich aus dem Datenbankentwurf (Abschnitt 5.2.3.1) und der Beschreibung der XML-Strukturkonstrukte (Abschnitt 5.2.3.2) zusammen. Das Verfahren der Umsetzung einiger Strukturkonstrukte wird in Abschnitt 5.2.4 dargelegt. Den Abschluß dieses Abschnitts bildet eine kurze Zusammenfassung.

5.2.1 Erfüllung der Kriterien durch den TextExtender

Um den Ausgangspunkt für die Konzeption festzulegen, soll zunächst die Frage geklärt werden, inwieweit die Methoden des TextExtenders den Anforderungskriterien genügen.

Kriterien für Änderungsoperationen

zu 1. *Update mit DTD:* Der TextExtender überprüft generell keine implizite strukturelle Integrität, so daß dieses Kriterium nicht erfüllt wird.

zu 2. *Update ohne DTD:* Der TextExtender überprüft generell keine implizite strukturelle Integrität, so daß dieses Kriterium nicht erfüllt wird.

zu 3. *Indexaktualisierung nach Update:* Die nach einer Änderungsoperation notwendige Aktualisierung des Indexes wird beim TextExtender nicht durch ein Datenbankereignis gesteuert. Der Triggermechanismus ist hier ein Zeitereignis, das aufgrund eines bei der Indexerzeugung festgelegten festen Intervalls ausgelöst wird. Je nach Länge des Intervalls kann damit nicht garantiert werden, daß der Datenbankzustand mit dem Zustand des Index harmoniert.

zu 4. *Insert mit DTD:* Der TextExtender überprüft generell keine implizite strukturelle Integrität, so daß dieses Kriterium nicht erfüllt wird.

zu 5. *Insert ohne DTD:* Der TextExtender überprüft generell keine implizite strukturelle Integrität, so daß dieses Kriterium nicht erfüllt wird.

zu 6. *Indexaktualisierung nach Insert:* Entsprechend der Argumentation zur *Indexaktualisierung nach Update* wird die Aktualisierung durch ein Zeitereignis gesteuert.

zu 7. *Indexaktualisierung nach Delete:* Entsprechend der Argumentation zur *Indexaktualisierung nach Update* wird die Aktualisierung durch ein Zeitereignis gesteuert.

Kriterien für Anfrageoperationen

zu 1. *Trennung Struktur/Inhalt:* Die einzige Strukturunterscheidung des TextExtenders besteht in der Trennung zwischen Worten, Sätzen und Paragraphen. Es erfolgt jedoch keine Differenzierung zwischen den in XML-Dokumenten enthaltenen Strukturelementen und Inhalten der Strukturelemente. Durch eine Modellierung der Strukturelemente als Attribute des objektrelationalen Datenbankschemas kann eine Trennung vorgenommen werden. Anfragen nach Strukturelementen werden dann auf Schemaanfragen abgebildet. Inhaltsanfragen entsprechen Anfragen auf Inhalte der strukturierten Attribute des Datenbankschemas.

zu 2. *Attributanfragen:* Über die Abstandsfunktion *IN SAME SENTENCE AS* kann nur ermittelt werden, ob sowohl der Attributname als auch dessen Wert gemeinsam in einem Satz auftreten. Gewährleistet ist dabei nicht, ob es sich tatsächlich um das gesuchte Attribut/Wert-Paar handelt, da Attribut/Wert-Paare und "normale" Sätze, die sowohl das Attribut als auch dessen Wert enthalten durch die Methode nicht unterschieden werden. Auch dieses Kriterium könnte über die Repräsentation der XML-Attribute als Attribute einer Relation realisiert werden. Eine Anfrage erfordert dann nicht einmal die Verwendung der Methoden des TextExtenders.

zu 3. *Bereichsauswahl:* Eine direkte Beschränkung der Anfrage auf bestimmte Teile des Dokumentes wird durch die Methoden des TextExtenders nicht unterstützt. Die Einschränkung kann nur indirekt, durch Aufsplittung des Dokumentes und separate Indizierung erfolgen.

zu 4. *Abstandsanfragen:* Die durch den TextExtender realisierbaren Abstandsfunktionen wurden bereits in Abschnitt 4.3.2.6 dargestellt. Die Möglichkeiten dieser Funktionen bieten keine direkte Unterstützung des Kriteriums. Eine indirekte Möglichkeit ist durch die Zerlegung der Dokumente gegeben.

zu 5. *Inklusionsbeziehungen:* Die fehlende Unterscheidung von Struktur und Inhalt verhindert Anfragemöglichkeiten auf genesteten Elementen mit Hilfe der Methoden des TextExtenders. Werden bei einer Zerlegung der semistrukturierten Dokumente die Inklusionsabhängigkeiten berücksichtigt, könnten diese Anfragen über das Datenbankschema realisiert werden.

zu 6. *Typisierung:* Die Information Retrieval-Methoden des TextExtenders sind rein textorientiert, so daß die Terme einer Anfrage generell als Zeichenketten interpretiert werden. Durch eine Modellierung der Inhalte der Elemente als Attribute von Relationen werden Vergleichsanfragen auf anderen Datentypen realisierbar. Dies setzt jedoch voraus, daß der Typ des Inhaltes in jedem semistrukturierten Dokument identisch ist.

zu 7. *Indizierung:* Wie bereits erwähnt, wird nicht zwischen Struktur und Inhalt differenziert. Folgerichtig werden Strukturelemente nur wie gewöhnlicher Text behandelt.

Als Fazit kann festgehalten werden, daß sich der TextExtender nicht auf direkte Weise für das Information-Retrieval semistrukturierter Dokumente eignet. Für fast alle Anforderungskriterien wurde als alternative Realisierung die Abbildung der DTD-Beschreibung auf das DB2-Datenmodell und damit die vollständige Zerlegung des Dokumentes in seine Strukturbestandteile angegeben.

Warum die Zerlegung des impliziten Schemas dennoch nicht die Basis der Konzeption bilden wird, soll im nachfolgenden Abschnitt dargelegt werden.

5.2.2 Probleme der Abbildung von XML-Strukturen auf Datenbankstrukturen

Angesichts der Forderung, daß XML-Dokumente eine der DTD-Beschreibungen des Anhang A entsprechende Struktur besitzen sollen, besteht prinzipiell die Möglichkeit ein Datenbankschema für diese Strukturen zu entwerfen. Die Zerlegung der impliziten Struktur der semistrukturierten Dokumente auf das Datenbankschema birgt jedoch Probleme, die nachfolgend beschrieben werden sollen.

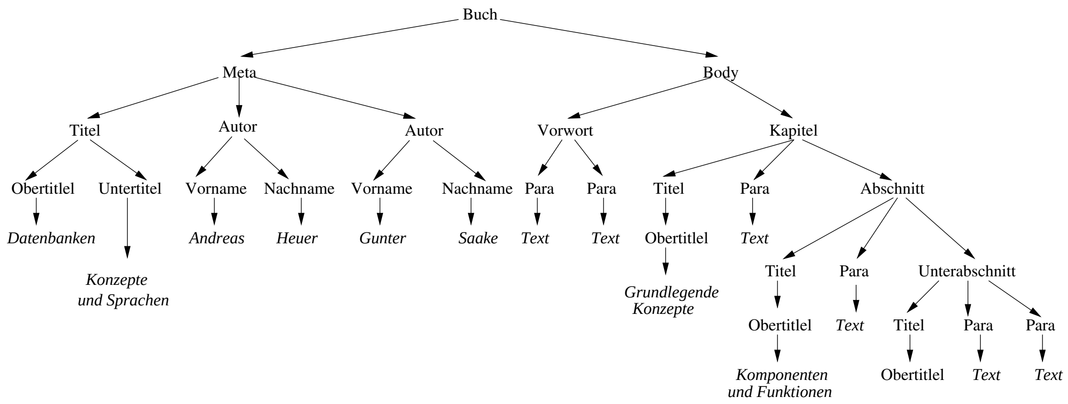

Abbildung 5.1: Baumstruktur eines XML-Dokumentes

Browsing-Anfragen sind schwer bzw. nicht realisierbar.

Wie bereits in Kapitel 3 dargestellt, besitzen XML-Dokumente eine Baumstruktur, wobei die Knoten des Baumes Strukturelemente sind und die Blätter Informationen enthalten, die diese Strukturelemente beschreiben (siehe Abbildung 5.1). Durch die Abbildung der DTD-Beschreibung auf das DB2-Datenmodell kann eine sehr gute Strukturbeschränkung auf solche Strukturelemente vorgenommen werden, die Vaterknoten eines Blattes sind, da sich Anfragen hier direkt auf ein einzelnes Attribut[2] beziehen. Beispielsweise kann die Anfrage

Gibt es Bücher, mit dem Haupttitel "objektrelationale Datenbanken"?

durch den folgenden SQL-Ausdruck realisiert werden, wenn *Titel* ein Attribut der Relation *Buch* ist.

```
SELECT docId
FROM   buch->titel->obertitel
WHERE  db2tx.contains(titlehandle, "objektrelationale Datenbanken" )=1
```

Bei Strukturbeschränkungen auf ein Strukturelement dicht dem Wurzelelement mit einer sehr tiefen Struktur (beispielsweise das Strukturelement *Kapitel* in der

[2] Attribut ist hier im Datenbanksinn zu verstehen

Abbildung 5.1) muß hingegen der gesamte Unterbaum dieses Knotens durchsucht werden. Dies kann nicht nur mehrere Attribute einer Relation, sondern auch mehrere Attribute verschiedener Relationen betreffen. Im schlechtesten Fall, wenn sich also die Anfrage auf das vollständige Dokument bezieht, muß das gesamte Datenbankschema in die Suche einbezogen werden.

Informationsverlust durch fehlende Typisierung.

Die im Rahmen dieser Arbeit zu verwaltenden und anzufragenden semistrukturierten Dokumente stellen Instanzen der in Anhang A aufgestellten DTDs dar. Die Nutzung von DTDs schränkt die strukturellen Möglichkeiten der Dokumente ein, so daß sich eine Abbildung auf das DB2-Datenmodell finden läßt. Seit der Version 5.2 sind hier auch Typhierarchien zulässig, so daß Alternativen innerhalb der Dokumentenbeschreibung auf Typhierarchien abgebildet werden können. Trotz der Abbildungsmöglichkeiten besteht das Problem der fehlenden Typisierung der Strukturelemente der DTD. Eine Dokumentenbeschreibung legt zwar die Namen der Elemente und Attribute sowie deren logische Schachtelung fest, es können aber keine Typfestlegungen für Attributwerte und Inhalte der Elemente getroffen werden. Durch die Erzeugung eines Datenbankschemas wird aber jedem Attribut ein fester Datentyp zugeordnet. Bei der Zerlegung der semistrukturierten Dokumente auf das Datenbankschema muß der Inhalt der Strukturelemente in diesen Zieldatentyp konvertiert werden können. Ist dies nicht möglich, droht an diesen Stellen Informationsverlust.

Keine Rekonstruktion des Originaldokumentes durchführbar.

Ein weiteres Problem, das durch die Aufsplittung der Dokumente zu Tage tritt, ist die mangelnde Rekonstruktionsmöglichkeit der Dokumente. Wurde der Inhalt eines Dokumentes erst einmal auf die Attribute des Datenbankschemas verteilt, ist keine Rekonstruktion des Originaldokumentes realisierbar. Es wäre daher notwendig, auch das Originaldokument in der Datenbank (bzw. im File-System) abzuspeichern, um Anfragen durchführen zu können, die eine Projektion auf das Dokument vorsehen. Dies bedeutet eine erhebliche Datenredundanz, da somit alle Dokumente dupliziert würden.

5.2.3 Konzeption

Aus den zuletzt genannten Gründen ist die Abbildung der gesamten DTD-Strukturen auf das objektrelationale Datenmodell nicht für das Information-Retrieval geeignet. In Kapitel 2 wurde bereits eine *hybride Lösung* vorgeschlagen. Diese sieht vor, nur *einige Strukturelemente* auf *Attribute des Datenbankschemas abzubilden.* Bei diesen Strukturelementen handelt es sich um *Metainformationen* der jeweiligen Objekttypen, die teilweise sogar obligatorisch sind. Da diese Elemente in der Regel eine höhere Anfragehäufigkeit aufweisen, kann die Beantwortung direkt auf Grundlage der Attribute erfolgen. Zusätzlich sieht das Datenbankschema ein Attribut des Datentyps *Character Large Object* (CLOB) vor, das der Aufnahme des Originaldokumentes dient. Sämtliche Information-Retrieval-Anfragen erfolgen dann über dieses Attribut bzw. den mit Hilfe des TextExtenders erzeugten Index. Um auch die implizite Struktur in Anfragen berücksichtigen zu können, muß die nutzerdefinierte *contains*-Funktion des TextExtenders erweitert werden, da sie, wie bereits gesehen, nur Textanfragen zuläßt. Diese Vorgehensweise hat den Vorteil, daß zum einen eine Rekonstruktion des Dokumentes entfällt und zum anderen Browsing-Anfragen direkt auf der Struktur des Dokumentes ausgeführt werden können. Bestehen bleibt das Problem, daß die Metainformationen unter Umständen nicht in den Zieldatentyp konvertiert werden können, was jedoch hier nicht zu Informationsverlust führt, da die Originalinformation weiterhin zur Verfügung steht.

Im weiteren Verlauf des Abschnitts wird zunächst das Datenbankschema für die Verwaltung von Dokumenten aus dem Bereich "Digitale Bibliotheken" vorgestellt. Anschließend erfolgt die Konzeption der Information-Retrieval-Methoden, die eine Erweiterung der *contains*-Methode darstellen.

5.2.3.1 Datenbankentwurf

Im konzeptionellen Datenbankentwurf wird eine Abgrenzung eines Teils der realen Welt vorgenommen, der den zu modellierenden Ausschnitt bestimmt. In der vorliegenden Diplomarbeit ist dieser Ausschnitt die "Digitale Bibliothek". Das Ergebnis ist zunächst die Darstellung des Datenbankschemas durch ein zielsystem-unabhängiges EER-Modell[3] (siehe Abbildung 5.2). Das Modell zeigt eine totale Partitionierung der Dokumente in die Objekttypen *Buch, Diplomarbeit, Doktorarbeit* und *Proceeding.* Die disjunkte Überdeckungsbedingung soll verhindern, daß Dokumente in die Datenbank aufgenommen werden, die zu keinem Spezialisierungstyp gehören. Die Aufnahme weiterer Dokumententypen erfordert die Erzeugung der entsprechenden Untertypen. Die in der DTD-Beschreibung *Buch* dargestellte Alternative zwischen *Autor* und *Editor* wurde für den Objekttyp *Buch* durch eine totale Partitionierung in die Untertypen *Buch_A* und *Buch_E* modelliert. Das Modell selbst enthält nur Metainformationen über Literatureinträge (siehe Abbildung 5.1 — Strukturelemente unterhalb des Meta-Elementes). Spezifische Informationen über Inhalte (siehe Abbildung 5.1 — Strukturelemente unterhalb des Body-Elementes) von Dokumenten wurden aus den zuvor genannten Gründen nicht in das Modell übernommen und müssen daher direkt aus dem Volltext des Dokumentes gewonnen werden.

Im nächsten Schritt wird das zielsystem-unabhängige Modell auf das objektrelationale Datenmodell von DB2 abgebildet. Es mag zunächst verwundern, warum für die Modellierung das EER-Modell verwendet wurde, obwohl in den Kapiteln

[3]EER - erweitertes Entity-Relationship-Modell

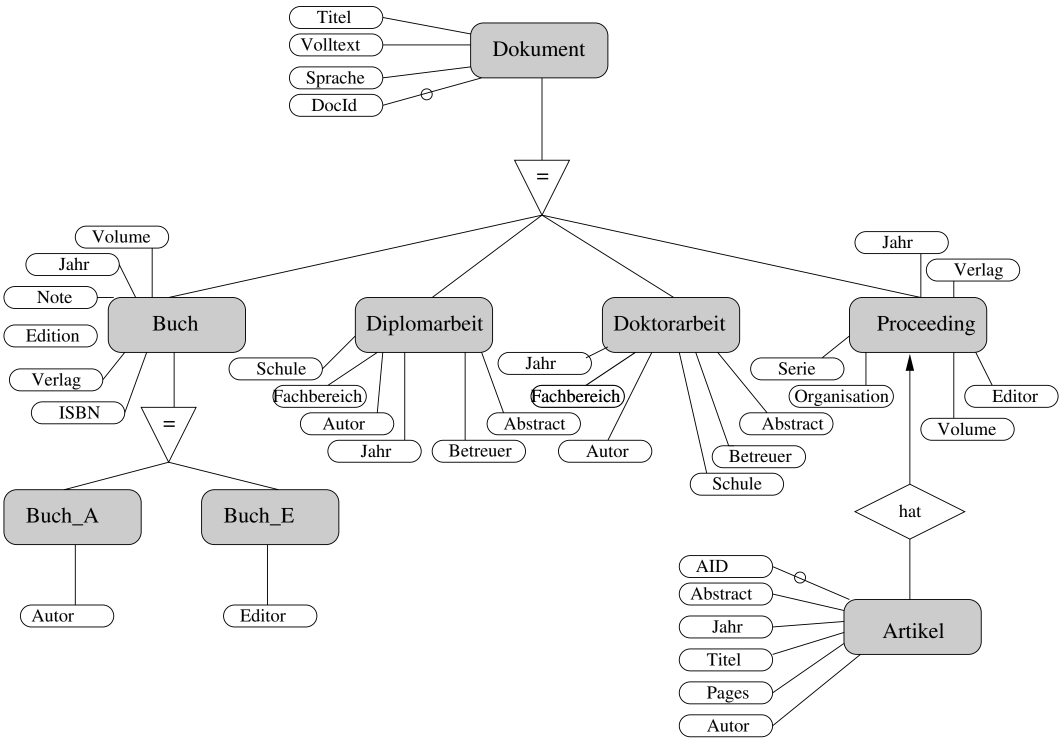

Abbildung 5.2: EER-Modell

1 bis 4 von einem weitestgehend relationalen Datenmodell von DB2 gesprochen wurde. Der Grund ist, daß während der Diplomarbeitsphase die DB2 Version 5.0 der Universal Database auf die Version 5.2 aktualisiert wurde. Seit dieser Version können nun im Datenbankschema auch strukturierte Typen und Typhierarchien vereinbart werden. Leider existierte keine Dokumentierung der objektorientierten Erweiterungen, so daß die neuen Möglichkeiten erst in der Endphase der Diplomarbeit erkannt werden konnten.

Die Abbildung vom EER-Modell auf das objektrelationale Datenmodell wird als *logischer Entwurf* bezeichnet. Dieser Entwurf besteht aus den beiden Schritten:

1. Transformation vom EER-Modell in das objektrelationale Datenbankmodell

2. Normalisierung des objektrelationalen Datenbankschemas

Das Ergebnis des logischen Entwurfs ist ein logisches Schema, welches nachfolgend angegeben ist. Auf die Angabe der Transformations- und Normalisierungsschritte wurde verzichtet. Es sei an dieser Stelle auf [HS95] verwiesen. Hier wird ein logischer Datenbankentwurf für die Abbildung eines ER-Modell auf ein relationales Modell ausführlich beschrieben.

Objektrelationales Schema

Für die Darstellung des objektrelationalen Datenbankschemas wird folgende Notation verwendet:

$$R = \{A_1, ..., A_n\} \text{ mit } n \in \text{IN}$$
$$S = \{R_1, ..., R_p\} \text{ mit } p \in \text{IN}$$

R ist ein Relationenschema, mit den Attributen $\{A_1, ..., A_n\}$. Die Schlüsselattribute der Relationenschemata werden durch Unterstreichung gekennzeichnet. Das objektrelationale Datenbankschema S ist dann eine Menge von Relationenschemata.

```
S =    {Titel, Person, Pages, Dokument, Buch, Diplomarbeit
       Doktorarbeit, Proceeding, Artikel, Proc_artikel,
       Editor, Autor, Betreuer}

Titel = { TitelId, Obertitel, Untertitel}
Person = {PersonId, PersonenTitel, Vorname, Middlename, Nachname}
Pages = {PageId, Von, Bis}
Dokument = {DocId, Sprache, TitelId, Volltext }
Buch = {Jahr, Verlag, ISBN, Note, Volume, Edition, Ersteller}
Diplomarbeit = {Schule, Fachbereich, Autor, Jahr, Abstract}
Doktorarbeit = {Schule, Fachbereich, Autor, Jahr, Abstract}
Proceeding = {Jahr, Verlag, Organisation, Serie, Volume}
Artikel = {AId, Jahr, TitelId, PagesId, Abstract, Proceeding}
Proc_artikel = {Proceeding, Artikel}
Editor = {EditorId, PersonId }
Autor = {AutorId, PersonId }
Betreuer = {BetrId, PersonId }
```

Für die Untertypen des Entity-Typs *Dokument* wurden nur die spezielleren Attribute angegeben. Durch die Modellierung als Typhierarchie werden die Attribute des jeweiligen Obertyps auf den Untertyp vererbt. Aus diesem Grund enthalten die Untertypen keine Kennzeichnung eines Schlüsselattributs, da das Schlüsselattribut *DocId* aus *Dokument* auf die Untertypen vererbt wird.

Ein gravierender Nachteil des DB2-Datenmodells sind die fehlenden Mengenkonstruktoren. Die Attribute *Autor* in *Buch_A* und *Artikel*, *Editor* in *Buch_E* und *Proceeding* sowie *Betreuer* in *Diplomarbeit* und *Doktorarbeit* stellen mengenwertige Attribute des Typs Person dar. Da Typkonstruktoren für mengenwertige Attribute jedoch nicht existieren, wurden diese mengenwertigen Attribute zu separaten Relationen (*Autor, Editor* und *Betreuer*) normalisiert. Auf die Umsetzung der totalen Partitionierung des Elementtyps *Buch* in die Typen *Buch_A* und *Buch_E* wurde ebenfalls verzichtet. Statt dessen wurde dem Entity-Typ *Buch* das Attribut *Ersteller* hinzugefügt, daß anzeigt, ob es sich beim Ersteller um Autoren oder Editoren handelt.

Fremdschlüsselbeziehungen

Fremdschlüssel stellen globale Integritätsbedingungen dar. Das heißt, daß sie über den Bereich einer Relation hinausragen. Durch die Fremdschlüsselbeziehungen werden demnach Abhängigkeiten zwischen Attributen verschiedener Relationen beschrieben.

Eine Fremdschlüsselbeziehung für eine Relation $r_1(R_1)$ ist ein Ausdruck

$$X(R_1) \longrightarrow Y(R_2)$$

mit $X \subseteq R_1, Y \subseteq R_2$. Die Attributliste X ist dann Fremdschlüssel in einem Relationenschema R_1, wenn in einem Relationenschema R_2 eine kompatible Attributliste Y Primärschlüssel ist.

```
TitelId(Dokument)         ⟶ TitelId(Titel)
AutorId(Autor)            ⟶ DocId(Buch)
AutorId(Autor)            ⟶ AId(Artikel)
EditorId(Editor)          ⟶ DocId(Buch)
EditorId(Editor)          ⟶ DocId(Proceeding)
BetrId(Betreuer)          ⟶ DocId(Diplomarbeit)
BetrId(Betreuer)          ⟶ DocId(Doktorarbeit)
Autor(Diplomarbeit)       ⟶ PersonId(Person)
Autor(Doktorarbeit)       ⟶ PersonId(Person)
PagesId(Artikel)          ⟶ PageId(Pages)
TitelId(Artikel)          ⟶ TitelId(Titel)
Proceeding(Proc_artikel)  ⟶ DocId(Proceeding)
Artikel(Proc_artikel)     ⟶ AId(Artikel)
```

5.2.3.2 Erweiterung des TextExtenders

Im Kapitel 4 wurden die recht umfangreichen Retrieval-Möglichkeiten des TextExtenders vorgestellt, die jedoch keine Differenzierung zwischen Strukturkonstrukten und Inhalten[4] vornehmen. Um eine Unterscheidung für die als Volltexte verwalteten semistrukturierten Dokumente treffen zu können, werden Konstrukte konzipiert, die die Syntax des TextExtender Suchargumentes auf das XML-Retrieval erweitern. Kernpunkt sollte dabei sein, daß die vollständige Syntax des TextExtenders (gemäß Anhang B) die Grundlage der Erweiterung bildet. Um

[4]Wenn von Inhalten gesprochen wird, sind Daten gemeint, die durch die Strukturkonstrukte näher beschrieben werden.

diese jedoch nutzen zu können, muß zumindest eine Funktion des TextExtenders Informationen über Textpositionen, der durch den Anfrageausdruck betroffenen Terme zurückliefern, um mit Hilfe dieser Koordinaten eine Unterscheidung zwischen Strukturelementen, Attributen und Inhalten vornehmen zu können. In Abschnitt 4.6 wurde jedoch bereits hervorgehoben, daß diese als *Highlighting* bezeichnete Funktion im TextExtender nicht zur Verfügung steht. Aus diesem Grund muß die Syntax des TextExtenders massiv eingeschränkt werden, um als Basis für die Erweiterung auf XML-Retrieval fungieren zu können. Im konkreten handelt es sich um die Funktionen *Synonym-Suche, Sound-Suche, Thesaurus-Erweiterung* und *Freitextsuche*. Theoretisch betrifft es auch die *Stammformsuche*, da diese in der Erweiterung nur als Teilstringsuche umgesetzt werden kann. Aus der Suchsyntax entfallen weiterhin die beiden Abstandsfunktionen des TextExtenders. Diese werden durch eine syntaktisch ähnliche Abstandsfunktion für XML-Strukturelemente ersetzt.

In den nachfolgenden Paragraphen werden Syntax und Semantik der Konstrukte beschrieben, die die reduzierte Syntax der TextExtender Suchanfrage auf das Retrieval semistrukturierter Dokumente erweitern. Diese XML-spezifischen Konstrukte können dann mit den verbliebenen Funktionen des TextExtenders kombiniert werden.

Einfache Strukturanfragen

In Kapitel 3 wurde beschrieben, daß ein XML-Dokument eine logische und eine physische Struktur besitzt. Durch die physische Struktur kann ein XML-Dokument aus verschiedenen *Entities* bestehen. Die logische Struktur teilt das Dokument in *ELEMENTE* ein. Die Erweiterung des Retrieval betrachtet hier nur die logischen Strukturkonstrukte, zu denen neben den *ELEMENTEN* noch die *ATTRIBUTE* zählen. Innerhalb der Elemente können neben weiteren Elementen die beschreibenden Daten enthalten sein. Diese Daten werden hier als *CONTENT* bezeichnet.

Nachfolgend werden die Konstrukte für einfache Strukturanfragen angegeben.

1. ELEMENT *elementname*

   Diese Funktion sucht im Dokument nach einem Strukturelement *elementname.*

   Beispiel: *Gibt es Dokumente, die ein Element Kapitel enthalten ?*
   SELECT docId
   FROM dokument
   WHERE contains(volltext, handle, 'ELEMENT "Kapitel" ')=1

2. ELEMENT *elementname1* IN *elementname2*

   Diese Funktion prüft, ob im Element *elementname2* das Strukturelement *elementname1* enthalten ist.

3. CONTENT *phrase*

   Diese Funktion sucht innerhalb der Strukturelemente nach dem Suchbegriff *phrase*.

4. CONTENT *phrase* IN *elementname*

   Diese Funktion sucht innerhalb des Strukturelementes *elementname* nach dem Suchbegriff *phrase*.

   Beispiel: *Gibt es Dokumente, die den Begriff Datenbanken im Titel enthalten ?*

   SELECT docId
   FROM dokument
   WHERE contains(volltext, handle,
   ' CONTENT "Datenbanken" IN "Titel" ')=1

5. ATTRIBUTE *attributename*

   Diese Funktion sucht im Dokument nach dem Attribut *attributename*.

6. ATTRIBUTE *attributename* IN *elementname*

   Diese Funktion prüft, ob im Strukturelement *elementname* ein Attribut *attributename* enthalten ist.

7. ATTRIBUTE *attributename* θ *value*
   $\theta \in \{<, >, =\}$

   Diese Funktion prüft, ob im Dokument ein Attribut *attributename* enthalten ist. Der Wert dieses Attributs wird mit Hilfe des Vergleichsoperators θ mit dem Wert *value* verglichen.

8. ATTRIBUTE *attributename* θ *value* IN *elementname*
   $\theta \in \{<, >, =\}$

   Diese Funktion prüft, ob im Strukturelement *elementname* ein Attribut *attributename* enthalten ist. Der Wert dieses Attributs wird mit Hilfe des Vergleichsoperators θ mit dem Wert *value* verglichen.

   Beispiel: *Gibt es Dokumente mit dem Element Buch, das ein Attribut/Wert-Paar Sprache=de enthält ?*

   SELECT docId
   FROM dokument
   WHERE contains(volltext, handle,
   ' ATTRIBUTE "Sprache"="de" IN "Buch" ')=1

Strukturanfragen mit Pfadausdrücken

In vielen Anwendungsfällen enthalten XML-Dokumente gleiche Strukturelemente mehrmals, so daß die Angabe eines Elementnamens im Suchausdruck nicht eindeutig ist. In anderen Situationen enthält ein Strukturelement viele weitere Elemente, die unter Umständen nicht bekannt sind, aber mit in die Suche einbezogen werden sollen. Anfragen, die diese Fälle berücksichtigen sollen, können durch einfache Strukturanfragen nicht bewältigt werden.

Durch die Angabe von Pfadausdrücken kann zum einen eine exaktere Lokalisierung eines Strukturelementes vorgenommen werden. In diesem Fall ist die gesamte Hierarchie von der Wurzel bis zum Zielelement anzugeben. Zum anderen kann die Suche durch Pfadausdrücke auf ganze Unterbäume ausgedehnt werden, wenn innerhalb des Pfadausdruckes statt eines Elementnamens ein Wiederholungsoperator verwendet wird. Um die Möglichkeiten der Pfadausdrücke zu verdeutlichen, werden nachfolgend die EBNF-Regeln angegeben, die die Syntax der Pfadausdrücke definieren. Die vollständige konzeptionelle Syntax findet sich in Anhang C.

```
PathExpression   ::= 'IN' PathExpression1
PathExpression1  ::= Elementname AttributeDef? |
                     Elementname AttributeDef? '.' PathExpression1 |
                     IterateOp '.' PathExpression1

AttributeDef     ::= 'WITH' AttributeValue
AttributeValue   ::= AttributeValue1 |
                     AttributeValue1 'AND' AttributeValue
AttributeValue1  ::= Attributename (Compare Value)? |

IterateOp        ::= '*' |
                     '+' |
                     '?'

Compare          ::= '>' |
                     '<' |
                     '='
```

Die beiden folgenden Beispiele sollen die Verwendung von Pfadausdrücken verdeutlichen. Durch den Ausdruck

*"Buch"."Body".**

wird ein XML-Dokument zunächst auf ein *Buch*-Element hin durchsucht. Dieses Element muß als direkten Nachfolger ein Element *Body* besitzen. Der eigentliche Suchbegriff kann dann im *Body*-Bereich in beliebiger Schachtelungstiefe gefunden werden. Zu jedem im Pfadausdruck spezifizierten Elementnamen können desweiteren Attribut/Wert-Paare angegeben werden, die dieses Element enthalten muß. Für den Pfadausdruck

*"Buch" WITH "Sprache"="de"."Body".**

gilt, daß der gesamte Unterbaum von *Body* mit durchsucht wird, falls das XML-Dokument ein Wurzelelement *Buch* enthält, das seinerseits ein Attribut/Wert-Paar *Sprache="de"* sowie ein Kindelement *Body* besitzt.

Die Konstrukte zu Strukturanfragen mit Pfadausdrücken werden nachfolgend beschrieben.

1. ELEMENT *elementname* IN *pathexpression*

   Diese Funktion prüft, ob das Elemente *elementname* im Pfadausdruck *pathexpression* enthalten ist.

2. CONTENT *phrase* IN *pathexpression*

   Diese Funktion sucht im Inhalt der durch den angegebenen Pfadausdruck *pathexpression* betroffenen Elemente nach dem Suchbegriff *phrase*.

3. ATTRIBUTE *attributename* IN *pathexpression*

   Diese Funktion prüft, ob in den durch den angegebenen Pfadausdruck *pathexpression* betroffenen Elementen ein Attribut *attributename* enthalten ist.

4. ATTRIBUTE *attributename* θ *value* IN *pathexpression*
   $\theta \in \{<, >, =\}$

   Diese Funktion prüft, ob in den durch den angegebenen Pfadausdruck *pathexpression* betroffenen Elementen ein Attribut *attributename* enthalten ist. Der Wert dieses Attributs wird entsprechend dem Vergleichsoperator θ mit dem Wert *value* verglichen.

Abstandsfunktionen

In Abstandsanfragen auf semistrukturierten Dokumenten ist das gemeinsame Auftreten von Termen im Inhalt eines Elementes oder im Unterbaum eines Strukturelementes von Interesse. An dieser Stelle könnte argumentiert werden, daß diese Funktionalität bereits durch die zuvor konzipierten Strukturanfragen in Kombination mit den booleschen Operatoren abgedeckt wird. Dies ist aber nur dann korrekt, wenn das Element oder der Pfadausdruck im XML-Dokument eindeutig ist. Tritt beispielsweise das Element *Titel* im Dokument mehrfach auf, so werden durch die Anfrage

SELECT docId
FROM dokument
WHERE contains(volltext, handle,
' CONTENT “Datenbank” IN “Titel” &
CONTENT “relational” IN “Titel” ')=1

auch Dokumente selektiert, bei denen die Terme in unterschiedlichen Titelelementen auftreten.

Wie bereits zu Beginn des Abschnitts beschrieben, werden die Abstandsfunktionen des TextExtenders durch die folgenden XML-Abstandsfunktionen ersetzt. Die in der Aufzählung angegebene Syntax von XML-Abstandsanfragen wurde etwas vereinfacht. Die exakten Regeln zur Definition von Abstandsanfragen sind Anhang C zu entnehmen.

Die Strukturkonstrukte für XML-Abstandsfunktionen werden nachfolgend aufgezählt.

1. TERMS *phrase* IN SAME ELEMENT AS *SearchFactor1*

   Durch diese Funktion wird geprüft, ob die Begriffe in *phrase* und die Begriffe in *SearchFactor1* zusammen in einem beliebigen Element des Dokumentes auftreten.

2. TERMS *phrase* IN *elementname* IN SAME ELEMENT AS *SearchFactor1*

   Durch diese Funktion wird geprüft, ob die Begriffe in *phrase* und die Begriffe in *SearchFactor1* zusammen in dem Element *elementname* auftreten.

3. TERMS *phrase* IN *pathexpression* IN SAME ELEMENT AS *SearchFactor1*

   Durch diese Funktion wird geprüft, ob die Begriffe in *phrase* und die Begriffe in *SearchFactor1* zusammen in dem durch *pathexpression* definierten Unterbaum auftreten. Als Demonstration der Funktion soll das Autor-Element der Buch-DTD aus Anhang A dienen, das durch folgende Notation beschrieben wird.

   ```
   <!ELEMENT Autor (Vorname, Middlename?, Nachname)>
   ```

   Da ein Buchdokument unter Umständen mehrere Autoren enthalten kann, ist die Anfrage
   Welche Bücher hat der Autor Andreas Heuer geschrieben ?
   folgendermaßen zu formulieren.

   SELECT Titel->Obertitel
   FROM Buch b, Autor a
   WHERE contains(volltext, handle,
   ' TERMS “Andreas IN “Buch”.”Autor”.*

IN SAME ELEMENT AS "Heuer" ')=1
AND b.docId = a.AutorID

4. ELEMENT *elementname1 direction*(k) *elementname2*
   $k \mid \{k \in \text{nat.Zahlen} \land k \geq 1\}$
   direction $\in$ {RIGHT, LEFT, TOP, DOWN}

   Diese Funktion prüft, ob das Element *elementname1* k-Elemente vom Element *elementname2* in der Richtung *direction* entfernt ist.

5. ELEMENT *elementname1* IN *elementname2 direction*(k) *elementname3*
   $k \mid \{k \in \text{nat.Zahlen} \land k \geq 1\}$
   direction $\in$ {RIGHT, LEFT, TOP, DOWN}

   Diese Funktion prüft, ob das Element *elementname1* im Inhalt des Elementes *elementname2* k-Elemente vom Element *elementname3* in der Richtung *direction* entfernt ist.

6. ELEMENT *elementname1* IN *pathexpression direction*(k) *elementname2*
   $k \mid \{k \in \text{nat.Zahlen} \land k \geq 1\}$
   direction $\in$ {RIGHT, LEFT, TOP, DOWN}

   Diese Funktion prüft, ob das Element *elementname1* k-Elemente vom Element *elementname2* in Richtung *direction* entfernt ist. Das Element *elementname1* muß dabei unterhalb des Pfadausdruck *pathexpression* auftreten.

7. CONTENT *phrase direction*(k) *elementname*
   $k \mid \{k \in \text{nat.Zahlen} \land k \geq 1\}$
   direction $\in$ {RIGHT, LEFT, TOP, DOWN}

   Diese Funktion ermittelt, ob der Term *phrase* im Inhalt eines Elementes existiert. Trifft dies zu, muß dieses Element k-Elemente vom Element *elementname* in Richtung *direction* entfernt liegen.

8. CONTENT *phrase* IN *elementname1 direction*(k) *elementname2*
   $k \mid \{k \in \text{nat.Zahlen} \land k \geq 1\}$
   direction $\in$ {RIGHT, LEFT, TOP, DOWN}

   Diese Funktion ermittelt, ob der Term *phrase* im Inhalt des Elementes *elementname1* existiert. Trifft dies zu, muß dieses Element k-Elemente vom Element *elementname2* in Richtung *direction* entfernt liegen.

9. CONTENT *phrase* IN *pathexpression direction*(k) *elementname*
   $k \mid \{k \in \text{nat.Zahlen} \land k \geq 1\}$
   direction $\in$ {RIGHT, LEFT, TOP, DOWN}

   Diese Funktion ermittelt, ob der Term *phrase* innerhalb des Pfadausdruckes *pathexpression* auftritt. Trifft dies zu, muß das Element in dem sich der

Term befindet k-Elemente vom Element *elementname* in Richtung *direction* entfernt liegen.

Inklusionsoperationen

Durch die Inklusionsoperationen können geforderte Vorkommenshäufigkeiten für Elemente und Terme angegeben werden. Bei der Verarbeitung einer entsprechenden Anfrage werden nur solche Dokumente selektiert, die mindestens die spezifizierte Anzahl Terme oder Strukturelemente enthalten.

Die Strukturkonstrukte für XML-Inklusionsoperationen werden nachfolgend aufgezählt.

1. CONTENT *phrase* COUNT(k)
   $k \mid \{k \in \text{nat.Zahlen} \wedge k \geq 1\}$

   Diese Funktion prüft, ob der Term *phrase* mindestens k-mal in einem beliebigen Element enthalten ist.

2. CONTENT *phrase* IN *elementname* COUNT(k)
   $k \mid \{k \in \text{nat.Zahlen} \wedge k \geq 1\}$

   Diese Funktion prüft, ob der Term *phrase* mindestens k-mal im Element *elementname* enthalten ist.

3. CONTENT *phrase* IN *pathexpression* COUNT(k)
   $k \mid \{k \in \text{nat.Zahlen} \wedge k \geq 1\}$

   Diese Funktion prüft, ob der Term *phrase* mindestens k-mal im Inhalt des durch den Pfadausdruck *pathexpression* definierten Unterbaums enthalten ist.

4. ELEMENT *elementname* COUNT(k)
   $k \mid \{k \in \text{nat.Zahlen} \wedge k \geq 1\}$

   Diese Funktion prüft, ob das Element *elementname* mindestens k-mal im Dokument enthalten ist.

5. ELEMENT *elementname1* IN *elementname2* COUNT(k)
   $k \mid \{k \in \text{nat.Zahlen} \wedge k \geq 1\}$

   Diese Funktion prüft, ob das Element *elementname1* mindestens k-mal im Element *elementname2* enthalten ist.

6. ELEMENT *elementname* IN *pathexpression* COUNT(k)
   $k \mid \{k \in \text{nat.Zahlen} \wedge k \geq 1\}$

   Diese Funktion prüft, ob das Element *elementname* mindestens k-mal im Pfadausdruck *pathexpression* enthalten ist.

Typisierung

Eine wichtige Eigenschaft von Datenbanksystemen ist ihre Fähigkeit, Vergleichsoperationen auf Attributwerten[5] durchführen zu können. Durch die reine Textorientierung und Fokussierung auf linguistische Fähigkeiten unterstützt der TextExtender keinerlei Vergleichsoperationen. Die reine Textorientierung ist aber für ein Retrieval auf XML-Dokumenten unzureichend. Möchte man beispielsweise an ein XML-Dokument über Buchinformationen eine Anfrage nach Buchpreisen unter 100,00 DM stellen, so werden bei einem Stringvergleich Preise von 90,00 DM als höher interpretiert. Die fehlende Typisierung der Dokumente kann durch eine dynamische oder eingeschränkt statische Form zumindest teilweise überwunden werden. Bei der dynamischen Typisierung werden der Anfragesprache Konstrukte zur Typisierung hinzugefügt. Bei der Verarbeitung werden die Inhalte der betroffenen Strukturelemente beziehungsweise die Attributwerte in den Zieldatentyp konvertiert. Hierbei ist es auf jeden Fall erforderlich, daß eine aufgrund eines Typfehlers ausgelöste Ausnahme behandelt wird und nicht zur Terminierung führt. Bei einer eingeschränkt statischen Typisierung wird in der Dokumentenbeschreibung jedem Strukturelement in seiner Attributliste ein Typattribut zugeordnet. Neben diesem Typattribut kann das Element weitere Attribute enthalten, für die ebenfalls ein Typattribut hinzugefügt wird. Die Unterscheidung der einzelnen Typattribute muß dann über den Namen erfolgen.

Beispiel:

```
<!ELEMENT Kapitel (#PCDATA) >
<!ATTLIST Kapitel Kapitel_Typ CDATA #FIXED “String”
                  ident ID #REQUIRED
                  ident_Typ CDATA #FIXED “Int”>
```

Das Beispiel definiert ein Element *Kapitel*. Der Typ des Kapitels wird durch das Attribut *Kapitel_ Typ* als Zeichenkette festgelegt. Das zusätzliche Attribut *ident* besitzt den Typ Integer, der durch das Attribut *ident_ Typ* bestimmt wird. Durch eine Erweiterung des Parsers kann die Typprüfung der Attributwerte und Inhalte der Strukturelemente während der Validierung des Dokumentes erfolgen.

Obwohl im Konzept dieser Diplomarbeit eine Validierung der zu verwaltenden semistrukturierten Dokumente hinsichtlich ihrer DTD erfolgt, wurde in der Konzeption die dynamische Typisierung vorgezogen, da sie auch Typisierungen wohlgeformter XML-Dokumenten zuläßt. Zudem werden durch die statische Typisierung die Terme der Anfrage generell in den Datentyp des Strukturelementes konvertiert, was bei der dynamischen Typisierung nur dann erfolgt, wenn eine explizite Typangabe im Suchausdruck vorliegt. Andernfalls werden die Terme

[5] In DB2 können die Vergleichsoperatoren $>$,$<$ und $=$ auf allen Datentypen mit Ausnahme der LOB-Datentypen ausgeführt werden

des Suchargumentes, die Inhalte der Strukturelemente und der Attributwerte als Zeichenketten interpretiert. Ein weiteres Problem der statischen Typisierung ist, daß alle Nicht-Typattribute als notwendig (*#REQUIRED*) deklariert werden müssen, da sich aufgrund der *#FIXED*-Vorgabe des zugehörigen Typattributs ein XML-Prozessor auch bei Fehlen dieses Attributs so verhält, als stünde es im Dokument.

Die Strukturkonstrukte zur Typisierung der Anfrageterme werden nachfolgend aufgezählt.

1. CONTENT θ *phrase* TYPE *type*
   $\theta \in \{<, >, =\}$
   type $\in$ {INT, REAL, DATE, STRING}

   Diese Funktion vergleicht die Inhalte der Strukturelemente über den Vergleichsoperator θ mit dem Suchterm *phrase*. Dabei werden sowohl der Suchterm als auch die Inhalte der Strukturelemente in den Datentyp *type* konvertiert.

2. CONTENT θ *phrase* TYPE *type* IN *elementname*
   $\theta \in \{<, >, =\}$
   type $\in$ {INT, REAL, DATE, STRING}

   Diese Funktion vergleicht den Inhalt des Strukturelementes *elementname* über den Vergleichsoperator θ mit dem Suchterm *phrase*. Dabei werden sowohl der Suchterm als auch der Inhalt des Strukturelementes in den Datentyp *type* konvertiert.

3. CONTENT θ *phrase* TYPE *type* IN *pathexpression*
   $\theta \in \{<, >, =\}$
   type $\in$ {INT, REAL, DATE, STRING}

   Diese Funktion vergleicht die Inhalte der Strukturelemente unterhalb des XML-Baumes, der durch den Pfadausdruck *pathexpression* bestimmt wird, über den Vergleichsoperator θ mit dem Suchterm *phrase*. Dabei werden sowohl der Suchterm als auch die Inhalte der Strukturelemente in den Datentyp *type* konvertiert.

4. ATTRIBUTE *attributename* θ *value* TYPE *type*
   $\theta \in \{<, >, =\}$
   type $\in$ {INT, REAL, DATE, STRING}

   Diese Funktion prüft, ob ein Attribut *attributename* existiert. Im Falle dessen werden der Attributwert dieses Attributs und der spezifizierte Wert *value* in den durch *type* vorgegebenen Datentyp konvertiert. Der Vergleich dieser beiden Werte wird über den Operator θ vorgenommen.

5. ATTRIBUTE *attributename* θ *value* TYPE *type* IN *elementname*
   $\theta \in \{<, >, =\}$
   type $\in$ {INT, REAL, DATE, STRING}

   Diese Funktion prüft, ob das Attribut *attributename* im Element *elementname* existiert. Im Falle dessen werden der Attributwert dieses Attributs und der spezifizierte Wert *value* in den durch *type* vorgegebenen Datentyp konvertiert. Der Vergleich dieser beiden Werte wird über den Operator θ vorgenommen.

6. ATTRIBUTE *attributename* θ *value* TYPE *type* IN *pathexpression*
   $\theta \in \{<, >, =\}$
   type $\in$ {INT, REAL, DATE, STRING}

   Diese Funktion prüft, ob das Attribut *attributename* in dem durch *pathexpression* definierten XML-Unterbaum enthalten ist. Im Falle dessen werden der Attributwert dieses Attributs und der spezifizierte Wert *value* in den durch *type* vorgegebenen Datentyp konvertiert. Der Vergleich dieser beiden Werte wird über den Operator θ vorgenommen.

Strukturextraktion

Die bisher konzipierten Konstrukte stellen eine Erweiterung des Suchargumentes der TextExtender *contains*-Funktion dar. Entsprechend ist das Ergebnis einer Anfrage auf der erweiterten Funktion ein boolescher Wert. Durch diese Funktionalität werden zwar gezielte Anfragen über Struktur und Inhalt ermöglicht, jedoch findet derzeit keine Extraktion von Teilen des Dokumentes statt.

Die Konzeption einer Strukturextraktions-Funktion *extract* auf den semistrukturierten Dokumenten verwendet ebenfalls die für die Erweiterung der *contains*-Funktion konzipierten Anfragekonstrukte (siehe auch Anhang C). Die Arbeitsweise dieser Funktion ist derart, daß zunächst der Anfrageausdruck auf Gültigkeit getestet wird (dies entspricht der Funktionsweise der *contains*-Funktion). Im Anschluß daran werden aus den Dokumenten, die die Prädikate des Anfrageausdrucks erfüllen, Strukturelemente extrahiert. Um welche Strukturelemente bzw. XML-Teilbäume es sich dabei handelt, wird entweder durch die in den Prädikaten enthaltenen Pfadausdrücke bzw. Strukturelemente bestimmt oder in Prädikaten ohne Pfadausdrücke bzw. Strukturelemente durch die Lokalisierung des Terms im Dokument. Da aufgrund der Anfrage ein Strukturelement mehr als einmal extrahiert werden kann, erfolgt abschließend eine Duplikateliminierung. Bei der Entfernung der Duplikate wird nicht nur die Gleichheit von Strukturelementen geprüft, sondern auch, ob ein Strukturelement oder Teilbaum in einem anderen Teilbaum enthalten ist. Einige Beispiele sollen die Funktionsweise der *extract*-Funktion verdeutlichen.

Beispiel 1:

extract(x & y)

Hier müssen sowohl das Prädikat x als auch das Prädikat y erfüllt sein, damit eine Strukturextraktion erfolgen kann.

Beispiel 2:

extract(x | y)

Hier muß entweder das Prädikat x oder das Prädikat y erfüllt sein, damit eine Strukturextraktion erfolgen kann. Sind x und y erfüllt, wird die Extraktion entsprechend des Prädikates x vorgenommen. Wird nur ein Prädikat erfüllt, erfolgt die Extraktion entsprechend des zutreffenden Prädikates.

Die Strukturextraktion mit Hilfe des Konzeptes der nutzerdefinierten Funktionen kann auf zwei unterschiedliche Arten erfolgen. Zum einen kann die Extraktion als skalare Funktion realisiert werden. Da durch einen Suchausdruck mehrere Strukturelemente ausgewählt werden können, erfolgt hier eine Konkatenation dieser Elemente. Zum anderen kann die Strukturextraktion durch eine Tabellenfunktion erfolgen. Alle extrahierten Strukturteile werden hier als Ergebnistupel zurückgegeben. Für die Ergebnisse beider Funktionen gilt jedoch, daß eine nachfolgende Retrieval-Operation aufgrund der fehlenden Indizierung nicht möglich ist.

Werden für die Realisierung der *contains*-Funktion ebenfalls die Konzepte der nutzerdefinierten Funktionen verwendet (diese Frage wird in Abschnitt 5.2.4.1 geklärt), so kann eine Kombination der Funktionen innerhalb einer SQL-Anweisung erfolgen. Ein Beispiel für eine kombinierte Anfrage ist:

Gib alle Büchertitel aus, die der Autor Andreas Heuer geschrieben hat.

```
SELECT  extract(volltext, handle, 'ELEMENT "Obertitel" IN "Buch"."Titel" ')
FROM    buch
WHERE   contains(volltext, handle,
        ' TERMS "Andreas" IN "Buch"."Autor".* IN SAME ELEMENT AS
        "Heuer" ')=1
```

Enthielte die *extract*-Funktion über eine Konjunktion zusätzlich das Prädikat der *contains*-Funktion, so würde sowohl eine Extraktion des Titels als auch des Autor-Elementes erfolgen.

Erzeugung einer Rangordnung

Sowohl das Getess-Projekt als auch das Global-Info-Projekt repräsentieren Suchmaschinen, die eine Nutzeranfrage auf Anfragen an verschiedene Datenquellen abbilden. In der Regel handelt es sich hier um Anfragen, die das Enthaltensein des Suchausdrucks im Dokument prüfen. Das Ergebnis dieser Anfragen muß dem Nutzer in einer Rangordnung präsentiert werden, die die Qualität der Ergebnisdokumente zu dem gestellten Anfrageausdruck widerspiegelt. Da an ein XML-Anfragesystem zusätzliche Kriterien hinsichtlich des Retrievals gestellt wurden, muß auch die Ranking-Funktion neue Kriterien berücksichtigen, die die Struktur mit einbeziehen.

Die Ranking-Funktion des TextExtenders bildet eine Rangordnung der Ergebnismenge durch das Verhältnis der Anzahl der Treffer zur Dokumentengröße (siehe Abschnitt 4.3.2). Die Erweiterung basiert wie der TextExtender auf dem Boolean-Retrieval-Modell, so daß das Ranking generell nur simuliert werden kann (siehe Abschnitt 4.1). Als Maß für den Ranking-Wert werden folgende Kriterien vorgeschlagen:

- Entspricht ein Anfrageterm dem vollständigen Inhalt eines Strukturelementes, so wird dies höher bewertet, als ein Anfrageterm, der nur einem Teil des Inhaltes entspricht.
- Treffer in Strukturelementen nah dem Wurzelelement werden höher bewertet als Treffer tief in der Baumstruktur. Dadurch sollen Metainformationen stärker gewichtet werden. Ähnliches könnte erreicht werden, indem überprüft wird, ob der Treffer in einem Strukturelement gelandet wurde, das als Attribut im Datenbankschema enthalten ist.
- Da die Retrieval-Erweiterung aus den in 5.2.3.2 genannten Gründen nur die Teilstring- und präzise Suche berücksichtigt, wird das exakte Vorkommen eines Terms stärker gewichtet als das Auftreten eines Teilstrings.
- In der Bewertung wird die Vorkommenshäufigkeit der Anfrageterme berücksichtigt.

Die Ranking-Funktion verwendet für das Suchargument die in Anhang C dargestellte Syntax. Wurden im Anfrageausdruck Suchpfade vorgegeben, so beziehen sich Treffer nur auf die Strukturelemente dieses Pfadausdruckes. Entsprechend der TextExtender Ranking-Funktion ist das Ergebnis dieser Funktion ein numerischer Wert.

5.2.3.3 Erfüllung der Kriterien nach der Konzeption

Zum Abschluß der Konzeption eines XML-Retrieval-Systems auf Basis des DB2 TextExtenders wird validiert, ob durch die Konzeption, die zu Beginn dieses Kapitels (Abschnitt 5.1) aufgestellten Anforderungskriterien abgedeckt werden.

Kriterien für Änderungsoperationen

zu 1. *Update mit DTD:* Die zu verwaltenden Volltexte sind urheberrechtlich geschützt, so daß Änderungsoperationen auf den Dokumenten strikt abgewiesen werden müssen. Somit haben nur das Löschen und das Einfügen von Dokumenten Einfluß auf den Datenbankzustand.

zu 2. *Update ohne DTD:* Für diesen Punkt gelten die Ausführungen von Punkt 1.

zu 3. *Indexaktualisierung nach Update:* Das XML-Retrieval-System verwendet den Index des TextExtenders, so daß dieses Kriterium nur in dem Maße berücksichtigt werden kann, wie es durch das TextExtender-System unterstützt wird.

zu 4. *Insert mit DTD:* Für die zu verwaltenden Objekttypen wurden Dokumentenbeschreibungen erzeugt. Entsprechend müssen einzufügende XML-Dokumente gültig hinsichtlich ihrer DTD sein. Die Selektion der Attributinhalte und die Validierung der Dokumente kann durch eine der beiden folgenden Varianten erfolgen.

1. **Variante:** Pro Attribut wird eine skalare UDF erzeugt, die den Inhalt des korrespondierenden Strukturelementes aus dem XML-Dokument ausliest. Die Validierung des Dokumentes wird durch einen Trigger gesteuert, der den Aufruf einer nutzerdefinierten Prozedur auslöst, sobald ein Volltextdokument in die Datenbank eingefügt wird.

   Vorteil:

   Es muß nicht für jedes Attribut eine UDF erzeugt werden, sondern nur für Attribute mit unterschiedlichen Datentypen.

   Mit Ausnahme der Änderung von Datentypen haben Änderungen an den Relationenschemata keine Auswirkung auf die UDFs.

   Nachteil:

   Die Prüfung der Konformität ist von der Selektion der Attributinhalte abgekoppelt, so daß nicht gesichert ist, daß der Inhalt der korrespondierenden Strukturelemente mit dem Inhalt der Attribute harmoniert.

   Das Einfügen eines XML-Dokumentes ist ineffizient, da für jedes Attribut des entsprechenden Objekttyps eine UDF aufgerufen wird, die ihrerseits einen Parse-Prozeß auf dem Dokument durchführt.

2. **Variante:** Für jeden Untertyp des Dokumententyps wird eine Tabellenfunktion erzeugt. Diese Funktion liefert genau ein Ergebnistupel mit den Datentypen der Attribute des Relationenschemas. Das Einfügen erfolgt dann über den Aufruf

   INSERT INTO table (attribute)
   SELECT attribute
   FROM table (udf(xml-dokument)) AS t

Während des Parse-Prozesses der Attributinhalte kann gleichzeitig die Konformität hinsichtlich der DTD geprüft werden.

Vorteil:

Durch die gemeinsame Validierung und Selektion ist gesichert, daß die Inhalte der Attribute mit den Inhalten der korrespondierenden Strukturelemente übereinstimmen.

Diese Variante ist effizienter, da die Validierung und Filterung in einem Parse-Vorgang durchgeführt werden können.

Nachteil:

Trotz der möglichen Validierung durch die Tabellenfunktion ist eine zusätzliche Prüfung erforderlich, die durch einen Trigger ausgelöst wird. Dies ist notwendig, da das Einfügen eines Dokumentes über eine Tabellenfunktion nicht vom ungeprüften Einfügen unterschieden werden kann.

Änderungen des Relationenschemas erfordern generell die Anpassung der Tabellenfunktion.

Für diese Konzeption wird die erste Variante vorgeschlagen, da sie unabhängig von speziellen Relationenschemata ist und dadurch mehr Flexibilität bietet. Die verschiedenen Selektionsfunktionen variieren nur in ihrem Rückgabewert, so daß das Auslesen der Information durch ein separates Modul erfolgen kann, das in allen UDFs verwendet wird. Innerhalb der einzelnen Funktion muß nur die Konvertierung in den Zieldatentyp erfolgen.

zu 5. *Insert ohne DTD:* Die Konzeption der "Digitalen Bibliothek" sieht die Verwendung spezieller Dokumentenbeschreibungen für die Objekttypen vor. Aus diesem Grund wird der Versuch, ein Dokument ohne Referenzierung einer dieser Dokumentenbeschreibungen einzufügen, abgelehnt.

zu 6. *Indexaktualisierung nach Insert:* Für diesen Punkt gelten die Ausführungen zu Punkt 3.

zu 7. *Indexaktualisierung nach Delete:* Für diesen Punkt gelten die Ausführungen zu Punkt 3.

Kriterien für Anfrageoperationen

zu 1. *Trennung Struktur/Inhalt:* Die Realisierung der Trennung zwischen Termen, die als Strukturelemente interpretiert werden sollen und Termen die einem Inhalt eines Elementes entsprechen, erfolgte in der Konzeption durch die Einführung der Parameter *ELEMENT* und *CONTENT*. Je nach Wahl des Parameter bezieht sich die Anfrage entweder auf den Inhalt des Elementes oder auf den Elementnamen.

zu 2. *Attributanfragen:* Der Zugriff auf Attribute von Strukturelementen wird in der Konzeption durch den Parameter *ATTRIBUTE* berücksichtigt. Die Attributsuche kann zusätzlich einen Vergleich des Attributwertes im Dokument mit dem Attributwert der Anfrage mit einschließen.

zu 3. *Bereichsauswahl:* Die Parametrisierung eines Terms kennzeichnet zunächst nur das XML-Strukturkonstrukt, das in der Suche betrachtet werden soll. Dabei wird jedoch völlig außer Acht gelassen, daß das semistrukturierte Dokument desweiteren eine logische Struktur besitzt. Diese logische Struktur wird in der Konzeption durch die Spezifizierung von Pfadausdrücken berücksichtigt. Diese Pfadausdrücke gewährleisten die Einschränkung des parametrisierten Terms auf einen speziellen Bereich der logischen Struktur.

zu 4. *Abstandsanfragen:* In der Konzeption wurden zwei Arten von Abstandsanfragen berücksichtigt. Zum einen können Anfragen formuliert werden, die das gemeinsame Auftreten von Termen im Inhalt von Strukturelementen prüfen. Diese Abstandsanfrage entspricht den Abstandsanfragen des TextExtenders, wobei hier die Besonderheiten von XML beachtet wurden. Zum anderen wurden in die Konzeption Abstandsanfragen von Elementen aufgenommen. Diese Abstandsanfragen beinhalten einen Richtungsanzeiger, der angibt wo das entfernte Elemente zu suchen ist.

zu 5. *Inklusionsbeziehungen:* Die Konzeption der Inklusionsbeziehungen berücksichtigt zum einen Anfragen nach der Vorkommenshäufigkeit von Termen in einem Strukturelement oder XML-Teilbaum. Zum anderen werden Anfragen unterstützt, die eine Überprüfung des Enthaltenseins von Elementen in anderen Elementen oder Teilbäumen vorsehen. In diesen Anfragen können ebenfalls Vorkommenshäufigkeiten spezifiziert werden.

zu 6. *Typisierung:* Die aktuelle XML-Spezifikation enthält keine Festlegungen über konkrete Datentypen der Strukturelemente. Um aber eine ähnliche Mächtigkeit der Vergleichsanfragen wie auf den strukturierten Attributen des Datenbankschemas zu erreichen, muß die Anfragesprache neben den Vergleichsoperatoren auch eine Typspezifikation des Anfrageterms zulassen. In der Konzeption wurde dieser Forderung Rechnung getragen, in dem jedem Term mittels des Parameter *TYPE* ein Datentyp zugewiesen werden kann, der vorgibt, wie der Term zu interpretieren ist.

zu 7. *Indizierung:* Die Konzeption beinhaltet die Erweiterung der inhaltsbasierten Suche auf die Berücksichtigung der impliziten Struktur. Die Erweiterung umfaßt jedoch keine zusätzliche Indizierung der Strukturelemente. Vielmehr wird der Index des TextExtenders verwendet, um eine Vorselektion der Dokumente zu erreichen. Um auszuschließen, daß dabei bereits Dokumente herausgefiltert werden, deren Strukturelemente Bezeichnungen von Stoppworten tragen, kann

- einerseits die Indizierung ohne Verwendung einer Stoppwortliste erfolgen. Dadurch wird sichergestellt, daß alle Begriffe des Dokumentes indiziert werden. Bedingt dadurch, wird aber der Index wesentlich vergrößert, was sich wiederum negativ auf die Performance auswirkt.
- Andererseits könnte für jeden Term der Anfrage getestet werden, ob er in der Stoppwortliste enthalten ist. Ist dies der Fall, wird dieser Term nicht im Suchausdruck zur Vorselektion verwendet. In dieser Variante ist zwar der Index nicht vergrößert, jedoch muß für jeden Term der Anfrage getestet werden, ob er in der Stoppwortliste enthalten ist.

Auf Grund des zu erwartenden großen Umfangs der XML-Dokumente der "Digitalen Bibliothek" wird die zweite Variante bevorzugt, zumal die Prüfung, ob Terme als Stoppworte auftreten nur einmal pro Anfrage durchzuführen ist.

5.2.3.4 Zusammenfassung der Konzeption

Anhand der in Abschnitt 5.1 aufgestellten Kriterien wurde zunächst die Eignung der TextExtender-Funktionalität in Bezug auf das Retrieval auf semistrukturierten Dokumenten bewertet. Im Ergebnis dieser Bewertung wurde konstatiert, daß der TextExtender für das XML-Retrieval nur dann geeignet ist, wenn die implizite Struktur der Dokumente als Attribute von Relationenschemata dargestellt wird. Gleichzeitig wurden aber auch die Probleme dieser Zerlegung aufgezeigt, die insbesondere in einem *Informationsverlust* durch fehlende Typisierung, der problematischen Realisierung von *Browsing-Anfragen* sowie der schwierigen *Rekonstruktion* des Originaldokumentes zu sehen sind. Aufgrund der recht umfangreichen Probleme wurde dieser Ansatz nicht weiterverfolgt. Statt dessen wurde eine *hybride Lösung* vorgeschlagen, die zum einen nur bestimmte Strukturelemente der Dokumentenbeschreibungen als Attribute von Relationenschemata darstellt. Diese Strukturelemente repräsentieren Metainformationen der Dokumente, deren Umsetzung durch folgende Charakteristik gerechtfertigt ist:

- Diese Strukturelemente treten aufgrund der Dokumentenbeschreibung in jedem XML-Dokument auf.
- Anfragen beziehen sich wesentlich häufiger auf diese Elemente.
- Der Informationsgehalt dieser Elemente ist in der Regel sehr hoch.

Zum anderen wird das Originaldokument mit in der Datenbank verwaltet. Da bei einem Retrieval auf den Originaldokumenten die implizite Struktur durch den TextExtender keine Beachtung erfährt, wurde eine Erweiterung des Suchausdrucks auf Basis der TextExtender-Syntax konzipiert. Diese zusätzlichen Strukturkonstrukte berücksichtigen nun die implizite Struktur, was unter anderem durch die Validierung der Anforderungskriterien hinsichtlich der Strukturkonstrukte gezeigt wurde.

5.2.4 Prototypische Implementierung

In der Konzeption wurde bisher relativ losgelöst von einer praktischen Realisierbarkeit die Erweiterung der Syntax des TextExtenders um Konstrukte zum XML-Retrieval vorgenommen. Um die Fähigkeiten und Grenzen des XML-Retrievals auf Basis des TextExtenders aufzuzeigen, wird im folgenden die prototypische Implementierung der konzipierten Konstrukte vorgestellt. Die Umsetzung wird in dieser Diplomarbeit nur anhand der *contains*-Funktion sowie der Konstrukte für *einfache Strukturanfragen, Strukturanfragen mit Pfadausdrücken* und *Abstandsanfragen* (für Terme) gezeigt.

5.2.4.1 Wahl der Implementierungsart

Vor der konkreten Umsetzung der Anfragekonstrukte des konzipierten XML-Retrieval-Systems steht die Frage, auf welche Art dieses System implementiert werden soll. Prinzipiell stehen hier zwei Varianten zur Auswahl.

- Die Erweiterung des TextExtenders auf das XML-Retrieval kann durch Funktionen bzw. Klassen in einer eigenen Applikation realisiert werden.

  Vorteil:
  Der TextExtender verfügt über ein Application Programming Interface, das neben den Suchfunktionen auch eine Browsing-Funktion bereitstellt, mittels derer die zutreffenden Begriffe des Dokumentes erkannt werden können. Dadurch besteht hier grundsätzlich die Möglichkeit, die gesamten linguistischen Suchfunktionen des TextExtenders auch innerhalb der XML-Erweiterung zu verwenden.

  Nachteil:
  Die konzipierten XML-Konstrukte repräsentieren Selektionsbedingungen an die XML-Dokumente. Um mittels eines Anwendungsprogramms ein XML-Retrieval durchführen zu können, müssen zunächst durch eine SFW-Anweisung XML-Dokumente vom Datenbank-Server zum Anwendungsprogramm auf einem Datenbank-Client übertragen werden (siehe Abbildung 5.3). Aufgrund des relativ großen Datenvolumens der XML-Dokumente, kann bereits die Übertragung zum beträchtlichen Leistungsengpaß werden (in der Abbildung symbolisiert durch 3 Pfeile vom DB-Server zum DB-Client). Die Entscheidung, welche Dokumente letztendlich die Selektionsbedingungen erfüllen, wird erst im Anwendungsprogramm getroffen.

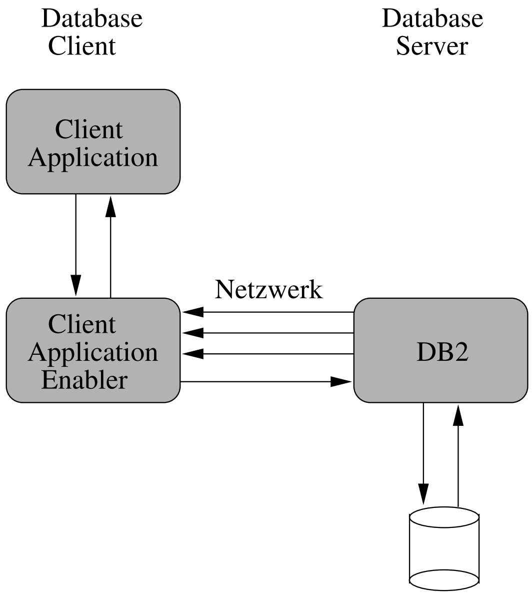

Abbildung 5.3: Selektion durch ein Anwendungsprogramm

- Die Realisierung des XML-Retrievals erfolgt mittels Funktionsaufrufen einer nutzerdefinierten Funktion (UDF).

  Vorteile:
  Die Selektion der Ergebnismenge einer XML-Retrieval-Anfrage erfolgt direkt auf dem Datenbank-Server durch eine UDF. An ein Anwendungsprogramm werden hier nur die XML-Dokumente übertragen, die den XML-Suchausdruck erfüllen (siehe Abbildung 5.4).

  Eine UDF entspricht einem SQL-Ausdruck, so daß eine UDF an solchen Stellen in SQL-Anweisungen auftreten darf, an denen SQL-Ausdrücke Gültigkeit besitzen. Durch die direkte Anwendbarkeit innerhalb einer SQL-Anweisung können nun XML-Retrieval-Anfragen in Programmen verwendet werden, die lediglich SQL-Anweisungen als Eingabe-Schnittstelle akzeptieren.

  Nachteil:
  Die zutreffenden Terme einer TextExtender-Suchanfrage können durch die UDFs des TextExtenders nicht ermittelt werden, so daß ein Großteil der linguistischen Suchfunktionalität des TextExtenders nicht auf die XML-Erweiterung angewendet werden kann (siehe Abschnitt 5.2.3.2).

Aufgrund der klaren Schnittstellendefinition und der besseren Performance wird für die Implementierung des XML-Retrieval-Systems die Verwendung nutzerdefinierten Funktionen vorgeschlagen. Die Nutzung des XML-Retrieval-Systems innerhalb der Projekte Getess und Global-Info setzt eine klare Schnittstellende-

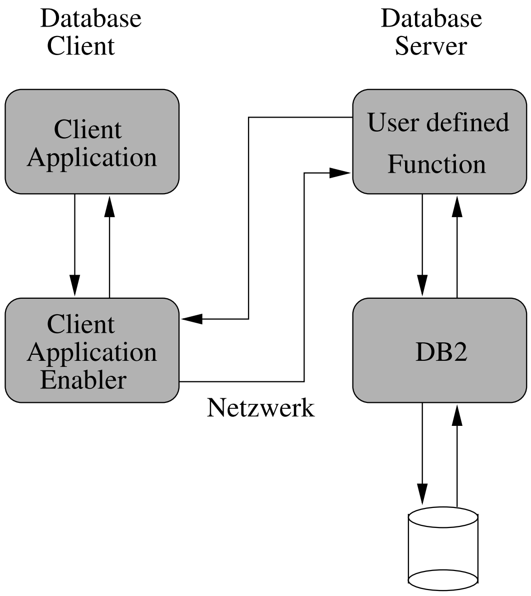

Abbildung 5.4: Selektion durch eine UDF

finition voraus, da hier eine Abbildung von einer Meta-Sprache (IRQL) auf eine Sprache eines Subsystems (DB2 SQL) vorgenommen wird. Die Einschränkungen hinsichtlich der linguistischen Funktionalität werden bewußt akzeptiert, jedoch nur unter dem Gesichtspunkt, daß durch die Diplomarbeit die Möglichkeiten und Grenzen der Erweiterung des TextExtenders aufgezeigt werden sollen.

5.2.4.2 Schnittstelle zwischen DB2 und der XML-contains-Funktion

Die Nutzung des Konzeptes der UDFs in DB2 impliziert zwei wesentliche Schritte. Zum einen muß die Funktionalität durch eine der unterstützten Programmiersprachen (C, C++, Java) umgesetzt werden. Bei der Realisierung muß die durch das Datenbanksystem aufzurufende Funktion Konformität zu der Schnittstellenbeschreibung von DB2 wahren. Zum anderen erfordert die Nutzung dieser Funktion die Bekanntmachung innerhalb des Datenbanksystems mit Hilfe des SQL-Kommandos *Create Function*.

Im weiteren Verlauf dieses Abschnitts wird zunächst die Schnittstellenbeschreibung zwischen dem DB2-System und einer UDF dargestellt. Anschließend werden die in der realisierten *contains*-Funktion verwendeten Argumente sowie die Bekanntmachung der Funktion durch die *Create Function*-Anweisung beschrieben. Aufgrund der Umsetzung mit Hilfe der Programmiersprache C wird die Beschreibung der Schnittstelle für diese Programmiersprache vorgenommen.

Wie bereits erwähnt, existiert zwischen dem Datenbanksystem DB2 und den nutzerdefinierten Funktionen eine klar definierte Schnittstelle [IBM97a]. Abbildung

5.5 zeigt die Argumente, die DB2 einer in C bzw. C++ realisierten Funktion übergibt. Alle auf der Achse liegenden Argumente sind obligatorisch und müssen dementsprechend auch in der Funktionsreferenz spezifiziert werden. Alle unterhalb der Achse befindlichen Argumente sind optional. Ein Rückpfeil deutet eine Wiederholungsmöglichkeit dieses Argumentes an. Die Übergabe der Argumente an die UDF sowie die Rückgabe der Ergebniswerte erfolgt mittels Zeigern.

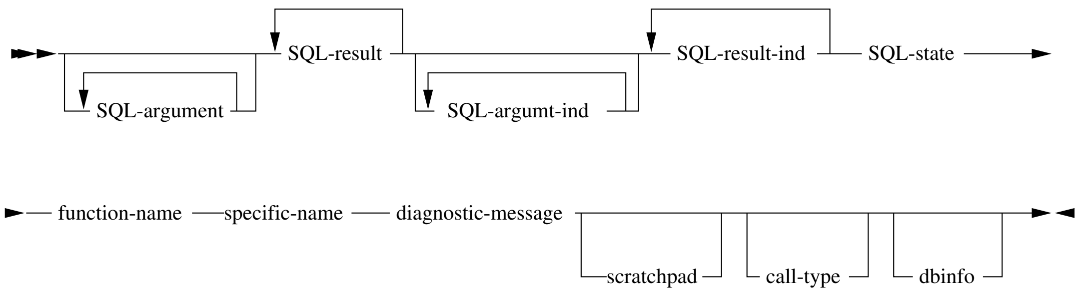

Abbildung 5.5: Argumente einer UDF [IBM97a]

Die Argumente sind folgendermaßen beschrieben[6]:

SQL-argument: Das Argument wird durch DB2 vor dem Aufruf der UDF gesetzt. Die Anzahl dieses Argumentes entspricht dabei der Anzahl Argumente in der Funktionsreferenz.

SQL-result: Dieses Argument muß durch die UDF vor dem Rücksprung zu DB2 gesetzt werden. Für skalare Funktionen existiert exakt ein SQL-result pro Funktionsaufruf. Für Tabellenfunktionen existiert ein SQL-result pro Ergebnistupel.

SQL-argument-ind: Das Argument wird durch DB2 vor dem Aufruf der UDF gesetzt. Es ist ein Indikator, der anzeigt, ob das korrespondierende SQL-argument einen Nullwert enthält oder nicht. Innerhalb der Argumentliste korrespondiert das n-te SQL-argument mit dem n-ten SQL-argument-ind. Der Indikator kann einen der beiden folgenden Werte enthalten.

0	Der Wert des korrespondierenden SQL-Argumentes ist nicht Null.
-1	Der Wert des korrespondierenden SQL-Argumentes ist Null.

SQL-result-ind: Dieses Argument muß durch die UDF vor dem Rücksprung zu DB2 gesetzt werden. Es ist ein Indikator, der anzeigt, ob der entsprechende Ergebniswert Null ist.

0 oder positiver Wert	Das Ergebnis ist nicht Null.
negativer Wert	Das Ergebnis ist ein Nullwert.

[6] Die exakte Beschreibung kann in [IBM97a] nachgelesen werden.

SQL-state: Dieses Argument kann durch die UDF vor dem Rücksprung zu DB2 gesetzt werden. Bei dem Argument handelt es sich um einen 5-Byte langen String, über den eine Fehlernummer an das Datenbanksystem zurückgegeben werden kann. Das Argument ist mit dem Wert '00000' initialisiert, was auf eine fehlerfreie Funktionsausführung hinweist.

function-name: Das Argument wird durch DB2 vor dem Aufruf der UDF gesetzt. Es enthält den qualifizierten Funktionsnamen (Schemaname und Funktionsname) der UDF im Datenbanksystem.

specific-name: Das Argument wird durch DB2 vor dem Aufruf der UDF gesetzt. Es enthält den spezifischen Funktionsnamen der UDF. Dieser kann entweder innerhalb des *Create Function*-Statements festgelegt werden oder er wird durch das Datenbanksystem generiert.

diagnostic-message: Dieses Argument kann durch die UDF vor dem Rücksprung zu DB2 gesetzt werden. Falls über das SQL-state Argument eine Fehlernummer an DB2 zurückgegeben werden soll, kann in diesem Argument eine Fehlerbeschreibung der Fehlernummer in *SQL-state* erfolgen.

scratchpad: Dieses Argument wird durch DB2 vor dem Aufruf der UDF gesetzt, falls das SCRATCHPAD-Schlüsselwort im *Create Function*-Statement spezifiziert wurde. Das *scratchpad* ist eine Struktur, in der die UDF Informationen speichern kann, die von einem Funktionsaufruf zum nächsten erhalten bleiben sollen. Diese Struktur wird beim ersten Aufruf der Funktion für eine SQL-Anweisung mit Null initialisiert und anschließend bis zum FINAL CALL-Aufruf durch DB2 nicht mehr modifiziert.

call-type: Dieses Argument wird durch DB2 vor dem Aufruf der UDF gesetzt, falls das FINAL CALL-Schlüsselwort im *Create Function*-Statement spezifiziert wurde. Für skalare Funktionen kann dieses Argument einen der drei folgenden Werte annehmen:

- -1 Es handelt sich um den *ersten Aufruf* der UDF für diese SQL-Anweisung. Alle Eingabeparameter werden der Funktion übergeben und ein vorhandenes *scratchpad* auf Null gesetzt.
- 0 Es handelt sich um einen *normalen Aufruf*. Alle Eingabeparameter werden der Funktion übergeben.
- 1 Es handelt sich um den *letzten Aufruf*. Die Eingabeparameter *SQL-argument* und *SQL-argument-ind* werden nicht an die Funktion übergeben.

dbinfo: Das Argument wird durch DB2 vor dem Aufruf der UDF gesetzt, falls das DBINFO-Schlüsselwort im *Create Function*-Statement spezifiziert wurde. Über dieses Argument werden der UDF diverse Informationen zur Verfügung gestellt, zu denen u.a. Informationen über die Datenbank Code Page, der Tabellenname auf den sich ein Update- oder Insert-Kommando bezieht, der Schemaname zu dieser Tabelle, der Spaltenname, die verwendete Plattform und die Datenbankversion zählen.

Nach der allgemeinen Beschreibung der Argumente einer UDF ist in der Tabelle 5.1 die konkrete Funktionsspezifikation der *contains*-Funktion angegeben. Die Reihenfolge der Parameter der Funktion entspricht der Reihenfolge in der Tabelle. Einige Parameter der UDF besitzen einen Datentyp, der nicht durch einen Standard-Datentyp in C repräsentiert werden kann. Im Software Development Kit (SDK) der UDB existiert jedoch eine Include-Datei (sqludf.h), in der alle in DB2 anwendbaren Datentypen sowie die Typen der UDF-Argumente durch C-Strukturen definiert sind. Alle mit "SQLUDF_" beginnenden Datentypen entstammen dieser Include-Datei.

DB2 Argumenttyp	C-Datentyp	Variablenname
SQL-argument	SQLUDF_CLOB	xmlData
SQL-argument	SQLUDF_VARCHAR_FBD	handleAttribute
SQL-argument	SQLUDF_LONG	searchArgument
SQL-result	short	resultValue
SQL-argument-ind	short	indXmlData
SQL-argument-ind	short	indHandleAttribute
SQL-argument-ind	short	indSearchArgument
SQL-result-ind	short	indResultValue
SQL-state	char	sqlState
function-name	char	functionName
specific-name	char	specificName
diagnostic-message	char	messageText
scratchpad	SQLUDF_SCRATCHPAD	scratchpad
call-type	SQLUDF_CALL_TYPE	calltype

Tabelle 5.1: Parameter der erweiterten *contains*-UDF

Die Erzeugung einer externen skalaren UDF erfordert neben der Realisierung dieser Funktion in einer von DB2 unterstützten Programmiersprache, die Bekanntgabe dieser Funktion innerhalb des Datenbanksystems durch die *Create Function*-Anweisung. Diese Anweisung besitzt eine recht umfangreiche Syntax, so daß hier nur die Klauseln erläutert werden, die für die *contains*-Funktion angewendet wurden. Die vollständig Syntaxbeschreibung kann in [Cha98] nachgelesen werden.

Die *Create Function*-Anweisung der erweiterten *contains*-Funktion hat folgende Gestalt:

```
CREATE FUNCTION contains
       (
             CLOB(2M)
             db2tx.DB2TextH
             LONG VARCHAR
       )
       RETURNS SMALLINT
       EXTERNAL NAME 'contains!contains'
       LANGUAGE C
       PARAMETER STYLE db2sql
       NOT VARIANT
       FENCED
       NOT NULL CALL
       NO SQL
       EXTERNAL ACTION
       SCRATCHPAD
       FINAL CALL
       ALLOW PARALLEL;
```

Die in dieser Anweisung verwendeten Klauseln besitzen folgende Bedeutung:

- *Eingabeparameter:* Die nach dem Funktionsnamen aufgeführten Datentypen legen die Typen der Funktionsparameter fest. Die *contains*-Funktion besitzt drei Parameter. Über den ersten Parameter vom Typ CLOB(2M) wird der Funktion ein XML-Dokument übergeben. Der zweite Parameter muß das Handle-Attribut dieses Dokumentes enthalten. Das Handle-Attribut wird durch die Indizierung des Dokumentes mit Hilfe des TextExtenders erzeugt. Der Datentyp dieses Parameters ist ein nutzerdefinierter Datentyp, der für den TextExtender erzeugt wurde. Der dritte Parameter enthält den XML-Suchausdruck.
- *RETURNS:* Diese Klausel legt den Datentyp fest, der durch die UDF aufgrund der Funktionsausführung zurückgegeben wird.
- *EXTERNAL NAME:* Diese Klausel identifiziert die Funktion als externe UDF. Gleichzeitig wird dem Datenbanksystem mitgeteilt, an welcher Stelle die Funktion gefunden werden kann. Die Funktion muß sich jedoch auf der Server-Maschine in einem Verzeichnis befinden, welches durch die Datenbank dynamisch geladen werden kann.
- *LANGUAGE:* Diese Klausel legt die Sprache fest, in der die externe UDF implementiert wurde. Im Falle der *contains*-Funktion ist hier C anzugeben.
- *PARAMETER STYLE:* Diese Klausel legt die Konventionen fest, die für die Parameterübergabe anzuwenden sind. Bei Verwendung der Programmiersprache C ist der Parameterstil *db2sql* zu spezifizieren.

- *NOT VARIANT:* Die NOT VARIANT-Klausel besagt, daß die Funktion keine unterschiedlichen Ergebnisse bei zwei Aufrufen der Funktion mit dem gleichen Parameter zurückliefern kann. Diese Klausel hat Einfluß auf den Optimierer, der in diesem Fall Anfragetransformationen durchführen kann.
- *FENCED:* Diese Option besagt, daß die Funktion in einem vom Datenbank-Manager getrennten Adreßraum ausgeführt wird. Dies führt zwar zu Leistungseinbußen, schützt jedoch die Datenbankintegrität vor Verletzungen aufgrund von Fehlern der Funktion. Erst nach der Testphase sollte eine Funktion in den *NOT FENCED*-Modus überführt werden.
- *NOT NULL CALL:* Diese Option weist das Datenbanksystem an, generell keine Nullwerte an die Funktion zu übergeben. Wird ein Nullwert durch das Datenbanksystem registriert, setzt das Datenbanksystem das Funktionsergebnis automatisch auf Null. Diese Option verbessert zum einen die Leistung, da für Nullwerte generell kein Funktionsaufruf erfolgt. Zum anderen ist ein Test auf Nullwerte innerhalb der Funktion hinfällig.
- *NO SQL:* Diese Klausel spezifiziert, daß die Funktion keine SQL-Anweisungen enthält. Zur Zeit sind in externen UDFs generell keine Zugriffe auf die Datenbank gestattet.
- *EXTERNAL ACTION:* Diese Klausel weist das Datenbanksystem darauf hin, daß die Funktion Seiteneffekte außerhalb des Datenbanksystems besitzt (beispielsweise werden Funktionsausschriften in einer Datei protokolliert). Der Datenbank-Optimierer führt aufgrund dieser Klausel keine Modifizierung der Anfrage hinsichtlich der Anzahl der Funktionsaufrufe durch.
- *SCRATCHPAD:* Durch diese Option wird der UDF ein Speicherbereich zugewiesen, in dem die Funktion Informationen von einem Funktionsaufruf zum nächsten bewahren kann. Im Fall der *contains*-Funktion wird das Scratchpad durch den TextExtender verwendet.
- *FINAL CALL:* Durch die Final-Call-Option wird die UDF einmal mehr als notwendig aufgerufen. In diesem letzten Funktionsaufruf kann dann der durch die Funktion allokierte Speicherplatz freigegeben werden. Das Datenbanksystem unterscheidet diesen letzten Aufruf von den anderen Aufrufen, da hier der Ergebniswert der Funktion keine Bewandtnis besitzt.
- *ALLOW PARALLEL:* Durch diese Option wird das Datenbanksystem legitimiert, eine parallele Funktionsausführung vorzunehmen.

5.2.4.3 Das XML-Retrieval-Modul

Die Realisierung der erweiterten *contains*-Funktion beinhaltet nicht allein die durch die *Create Function*-Anweisung dem Datenbanksystem bekanntgegebene Funktion, sondern ein komplettes Modul. Dieses Modul umfaßt

- einen *Parser*, der den Suchausdruck analysiert,
- die *TextExtender contains*-Funktion, die eine Vorfilterung der Dokumente realisiert,
- einen *Wrapper*, der aus den XML-Dokumenten eine Baumstruktur aufbaut
- sowie *diverse Funktionen*, die die Suche der Terme auf der Baumstruktur realisieren.

Die allgemeine Funktionsweise dieses Moduls wird durch Abbildung 5.6 widergespiegelt. Der Aufruf einer XML-Retrieval-Anfrage innerhalb einer SQL-Anwei-

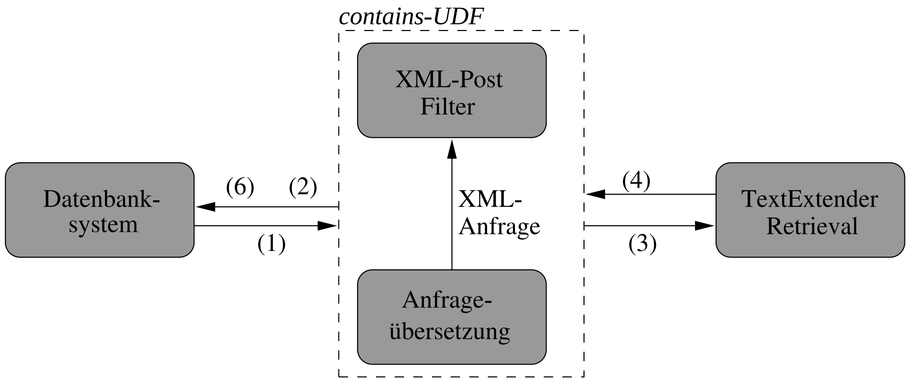

Abbildung 5.6: Funktionsweise des XML-Retrieval-Moduls

sung in DB2 hat zunächst den Aufruf der *contains*-UDF zur Folge (1). Der an die Funktion übergebene Suchausdruck wird mittels des Parser-Moduls hinsichtlich der Syntax des Anhangs C analysiert (Anfrageübersetzung). Ein fehlerhafter Suchausdruck beendet den Funktionsdurchlauf und übergibt eine Fehlermeldung (über die Parameter *sqlState* und *messageText*) an das Datenbanksystem (2). Ein fehlerfreier Suchausdruck wird in einen TextExtender-konformen Suchausdruck sowie in einen Binärbaum transformiert. Mit Hilfe des erzeugten TextExtender Suchausdrucks sowie dem Parameter *handleAttribute* wird eine Volltext-Retrieval-Anfrage an die TextExtender *contains*-Funktion gestellt (3). Diese Funktion (TextExtender Retrieval) vollführt eine Vorselektion der Dokumente. Dies bedeutet, daß für Dokumente, die nicht das TextExtender Suchargument erfüllen der anschließende XML-Retrieval-Durchlauf entfällt. Aufgrund des booleschen Ergebnisses der Funktion ist das Funktionsergebnis für diese Dokumente *false* (6). Die Vorselektion soll die Dokumentenmenge, auf die das XML-Retrieval angewendet wird, einschränken. Dies ist notwendig, da die XML-Erweiterung selbst über keinen eigenen Index verfügt, die direkte Suche auf den semistrukturierten Dokumenten aber ein wesentlich schlechteres Zeitverhalten gegenüber einer Indexsuche aufweist. Das abschließende XML-Retrieval auf den Dokumenten, die das TextExtender-Suchargument erfüllen, wird durch den *XML-Postfilter* durchgeführt. Das Ergebnis des XML-Postfilters wird anschließend an das Datenbanksystem zurückgegeben (6). Die Funktionsweise des XML-Postfilters ist in Abbildung 5.7 dargestellt.

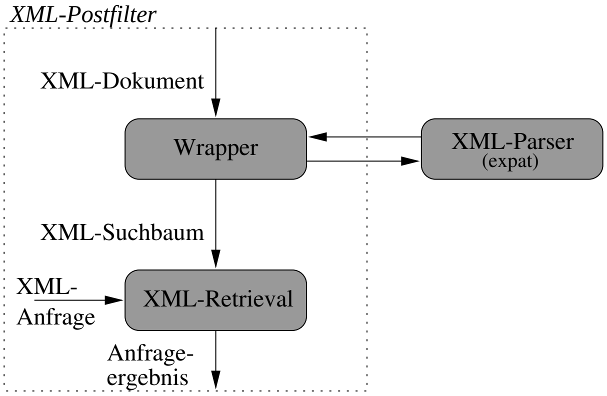

Abbildung 5.7: Funktionsweise der XML-Postfilterung

In Abschnitt 2.3 wurde für die XML-Erweiterung ein hybrides Datenmodell vorgeschlagen, in dem das XML-Retrieval auf einem Baummodell ausgeführt wird. Zum Zeitpunkt der Anfragestellung existieren jedoch nur XML-Dokumente, die während der Anfragebearbeitung in das Zielmodell transformiert werden müssen. Im XML-Retrieval-Modul erfolgt diese Transformation durch einen *Wrapper*, der ein XML-Dokument liest und in seine Strukturbestandteile zerlegt. Die einzelnen Strukturbestandteile werden in die Baumstruktur eingefügt. Für das Lesen und die Strukturextraktion von XML-Dokumenten wurde in der vorliegenden Diplomarbeit auf die Funktionalität des *expat*-Parsers[7] von James Clark zurückgegriffen. Die einzelnen Strukturbestandteile werden dann in die nachfolgend dargestellte XML-Baumstruktur (XML-Suchbaum) eingefügt.

```
struct XML-Tree {
       string        elementName;
       string        content;
       attributeList listOfAttributes;
       xmlList       childList;
       XML-Tree      parent };

struct xmlList {
       XML-Tree      childNode;
       xmlList       next };
```

[7]http://www.jclark.com/xml/expat.html

```
struct attributeList {
       string        attributeName;
       string        attributeValue;
       attributeList next };
```

Die Auswertung der Prädikate des Suchausdrucks (XML-Retrieval) wird auf der erzeugten Baumstruktur des Dokumentes vorgenommen. Hinzugefügt werden muß, daß Prädikate des Suchausdrucks, die keine Pfadangaben enthalten (einfache Strukturanfragen, Abstandsanfragen ohne Pfadangaben) aus Performance-Gründen bereits während der Baumerzeugung ausgewertet werden. Das Ergebnis des Funktionsaufrufs wird gebildet, indem die Ergebnisse der Prädikate über die logischen Operatoren des Suchausdrucks verknüpft werden.

5.2.5 Zusammenfassung

Die beispielhafte Implementierung eines Teils der Konzeption hat gezeigt, daß die Erweiterung des TextExtenders auf das XML-Retrieval prinzipiell möglich ist. Der Ausbau dieses Retrieval-Systems sowie die Verwendung unter realen Bedingungen (große Dokumentengröße und -anzahl) wird aufgrund der folgenden Probleme bzw. Einschränkungen aber nicht empfohlen:

- Aufgrund der fehlenden Lokalisierung der zutreffenden Terme geht der überwiegende Teil der linguistischen Funktionalität verloren.
- Das Zeitverhalten aufgrund des sequentiellen Lesens der XML-Dokumente zur Transformation in das Baummodell sowie der Auswertung der Prädikate des Suchausdrucks durch massiv-rekursiven Baumdurchlauf ist inakzeptabel.
- Nicht erfaßt werden konnte bisher, wie sich die Baumstruktur sowie die massiv-rekursiven Funktionen zum Durchlauf der Baumstruktur bei sehr tief geschachtelten XML-Strukturen verhalten.

Ein denkbarer Lösungsansatz um ein Retrieval auf semistrukturierten Dokumenten bezüglich Struktur und Inhalten durchführen zu können besteht in der Verwendung neuer Indizierungsmodelle, die bei der Indizierung die Struktur berücksichtigen. Dieser Ansatz wird beispielsweise in [Dao98] verfolgt.

5.3 Konzeption für Informix Universal Server

In Anbetracht der ausführlichen Darstellung der konzeptionellen Erweiterung des TextExtenders wird in der konzeptionellen Erweiterung des Excalibur DataBlades (respektive des Informix Universal Servers) nur auf Unterschiede zwischen Informix und DB2 eingegangen.

5.3.1 Erfüllung der Kriterien durch Excalibur

Entsprechend Abschnitt 5.2.1 wird zunächst der Ausgangspunkt der Konzeption bestimmt. Dazu werden die Methoden des Excalibur DataBlades hinsichtlich der Anforderungskriterien aus Abschnitt 5.1 validiert.

Kriterien für Änderungsoperationen

zu 1. *Update mit DTD:* Excalibur überprüft generell keine implizite strukturelle Integrität, so daß dieses Kriterium nicht erfüllt wird.

zu 2. *Update ohne DTD:* Excalibur überprüft generell keine implizite strukturelle Integrität, so daß dieses Kriterium nicht erfüllt wird.

zu 3. *Indexaktualisierung nach Update:* Im Unterschied zum TextExtender wird die Aktualisierung des Index durch ein Datenbankereignis gesteuert, so daß der Zustand des Index stets mit dem Datenbankzustand harmoniert.

zu 4. *Insert mit DTD:* Excalibur überprüft generell keine implizite strukturelle Integrität, so daß dieses Kriterium nicht erfüllt wird.

zu 5. *Insert ohne DTD:* Excalibur überprüft generell keine implizite strukturelle Integrität, so daß dieses Kriterium nicht erfüllt wird.

zu 6. *Indexaktualisierung nach Insert:* Die Aktualisierung des Index wird durch ein Datenbankereignis gesteuert (siehe *Indexaktualisierung nach Update*).

zu 7. *Indexaktualisierung nach Delete:* Die Aktualisierung des Index wird durch ein Datenbankereignis gesteuert (siehe *Indexaktualisierung nach Update*).

Kriterien für Anfrageoperationen

zu 1. *Trennung Struktur/Inhalt:* Die einzige Strukturunterscheidung des Excalibur DataBlades besteht in der Trennung von Worten. Entsprechend des TextExtenders erfolgt auch hier keine Differenzierung zwischen den in XML-Dokumenten enthaltenen Strukturelementen und Inhalten der Strukturelemente, so daß für die Erfüllung dieses Kriteriums die selben Bedingungen wie für den TextExtender gelten.

zu 2. *Attributanfragen:* Diese Anfrageart könnte durch die Proximity-Suche simuliert werden, indem der Attributname und dessen Wert eine maximale Distanz von drei Worten aufweisen. Ob diese Herangehensweise aber in jedem Fall korrekt abläuft, muß angezweifelt werden, da beispielsweise *Attribut=Wert* ohne Leerzeichen als ein Wort interpretiert wird. Wie beim TextExtender kann dieses Kriterium über die Repräsentation der XML-Attribute als Attribute einer Relation realisiert werden.

zu 3. *Bereichsauswahl:* Für die Bereichsauswahl gelten die selben Ausführungen wie für die Bereichsauswahl des TextExtenders in Abschnitt 5.2.1.

zu 4. *Abstandsanfragen:* In Excalibur formulierte Abstandsanfragen erwarten einen Parameter, der die maximale Distanz der Terme zueinander vorgibt. Dabei wird jedoch stets das Dokument als globaler Bereich herangezogen. Somit gilt, wie beim TextExtender, daß die Excalibur Abstandsanfragen keine direkte Unterstützung des Kriteriums bieten. Eine indirekte Möglichkeit ist durch die Zerlegung der Dokumente gegeben.

zu 5. *Inklusionsbeziehungen:* Für Inklusionsbeziehungen gelten die selben Ausführungen wie für Inklusionsbeziehungen des TextExtenders in Abschnitt 5.2.1.

zu 6. *Typisierung:* Für Typisierungen gelten die selben Ausführungen wie für Typisierungen des TextExtenders in Abschnitt 5.2.1.

zu 7. *Indizierung:* Für die Indizierung gelten die selben Ausführungen wie für die Indizierung des TextExtenders in Abschnitt 5.2.1.

Das Excalibur DataBlade ist gleichermaßen wie der TextExtender nicht für das Information-Retrieval semistrukturierter Dokumente geeignet. Auch die indirekten Nutzungsmöglichkeiten durch die Abbildung der XML-Strukturen auf die Datenbankstrukturen wirft die in Abschnitt 5.2.2 des TextExtenders diskutierten Probleme auf.

5.3.2 Konzeption

Die aufgeführten Probleme sind eine Indikation für die Nutzung des hybriden Ansatzes (siehe Abschnitt 2.3), in dem nur Meta-Informationen auf die Konzepte des Datenbankschemas abgebildet werden. Information-Retrieval-Anfragen werden indessen direkt auf dem als CLOB gespeicherten semistrukturierten Dokument ausgeführt. Wie beim TextExtender muß zur Beachtung der impliziten Struktur der *etx_contains*-Operator des Excalibur DataBlades erweitert werden. In Abschnitt 4.5.3 wurden die Restriktionen des DataBlades ganannt, worunter u.a. die Fixierung sämtlicher Tuning-Parameter auf das gesamte Suchkriterium sowie die nicht kombinierbaren *etx_contains*-Operatoren innerhalb einer WHERE-Klausel fallen. Angesichts dieser Einschränkungen erweist sich das DataBlade (respektive das Suchargument des DataBlades) als eher ungeeignet für die Skalierung auf XML-Retrieval. Um aber den Unterschied zum TextExtender zu zeigen, werden dennoch die Tuning-Parameter des *etx_contains*-Operators um XML-spezifische Parameter ergänzt.

5.3.2.1 Datenbankentwurf

Für die Excalibur-Erweiterung wird der in Abschnitt 5.2.3.1 vorgestellte konzeptionelle Datenbankentwurf (Abbildung 5.2) verwendet. Die Umsetzung des zielsystem-unabhängigen Modells auf das objektrelationale Datenmodell von Informix gestaltet sich aber einfacher als bei DB2, da Informix, wie bereits in Abschnitt 4.4 erwähnt, über mehr objektorientierte Konzepte verfügt. Zu diesen Konzepten gehören u.a. Komponenten- und Collection-Datentypen (*row type, set, multiset, list*).

Objektrelationales Schema

Die Darstellung des objektrelationalen Datenbankschemas erfolgt gemäß der in Abschnitt 5.2.3.1 verwendeten Notation:

$$R = \{A_1, ..., A_n\} \text{ mit } n \in \mathrm{IN}$$
$$S = \{R_1, ..., R_p\} \text{ mit } p \in \mathrm{IN}$$

```
S =    {Dokument, Buch, Buch_A, Buch_E, Diplomarbeit
        Doktorarbeit, Proceeding, Artikel, Proc_artikel}

Dokument = {DocId, Sprache, Titel, Volltext }
Buch = {Jahr, Verlag, ISBN, Note, Volume, Edition}
Diplomarbeit = {Schule, Fachbereich, Autor, Jahr, Abstract,
                Betreuer}
Doktorarbeit = {Schule, Fachbereich, Autor, Jahr, Abstract,
                Betreuer}
Proceeding = {Jahr, Verlag, Organisation, Serie, Volume, Editor}
Artikel = {AId, Jahr, Titel, Pages, Abstract, Pages, Autor,
           Proceeding}
Buch_A = {Autor}
Buch_E = {Editor}
Proc_artikel = {Proceeding, Artikel}
```

In den Untertypen von *Dokument* wurden wiederum nur die spezielleren Attribute aufgeführt, da die Attribute des Obertyps auf den Untertyp vererbt werden. Für die Komponenten-Typen *Pages, Titel* und *Personen* (im EER-Modell als *Autor, Editor* oder *Betreuer* benannt) werden jeweils Komponenten-Typen (*row types*) erzeugt. Die mengenwertigen Attribute *Autor* in *Buch_A* und *Artikel*, *Editor* in *Buch_E* und *Proceeding* und *Betreuer* in *Diplomarbeit* und *Doktorarbeit* können in Informix direkt über den SET-Konstruktor modelliert werden und müssen nicht, wie in DB2, zu separaten Relationen normalisiert werden.

Fremdschlüsselbeziehungen

Der überwiegende Teil der Fremdschlüsselbeziehungen des DB2 Datenbankentwurfs resultierte aus der Normalisierung der mengenwertigen Attribute. Da diese Attribute in Informix einen Collection-Datentyp annehmen können, entfallen diese Fremdschlüsselbeziehungen. Wie in DB2 ist eine Fremdschlüsselbeziehung für eine Relation $r_1(R_1)$ ein Ausdruck

$$X(R_1) \longrightarrow Y(R_2)$$

mit $X \subseteq R_1, Y \subseteq R_2$. Die Attributliste X ist dann Fremdschlüssel in einem Relationenschema R_1, wenn in einem Relationenschema R_2 eine kompatible Attributliste Y Primärschlüssel ist (siehe Abschnitt 5.2.3.1).

```
Proceeding(Proc_artikel) ⟶ DocId(Proceeding)
Artikel(Proc_artikel)    ⟶ AId(Artikel)
```

5.3.2.2 Erweiterung des Excalibur DataBlades

Analog zu Abschnitt 5.2.3.2 wird in diesem Abschnitt eine Erweiterung des Excalibur DataBlades konzipiert, die ein für XML-Dokumente spezifisches Retrieval erlaubt. Das Fundament bilden die Tuning-Parameter des *etx_contains*-Operators. Zur Illustration wird noch einmal der prinzipielle Aufbau dieses Operators gezeigt (siehe auch Abschnitt 4.5.2).

```
etx_contains(column, row(Suchkriterium, Tuning-Parameter), SLV)
```

Durch *column* wird dem Operator das Datenbankattribut mitgeteilt, auf dem eine Volltextsuche ausgeführt werden soll. Das zweite Argument ist ein Komponententyp, der im Suchkriterium die zu suchenden Terme enthält, die durch die Tuning-Parameter beeinflußt werden. Nachteil dieser Herangehensweise ist, daß sich die Wirkung der Tuning-Parameter eines Operators generell auf alle Terme des Suchkriteriums bezieht.

Der erweiterte *contains*-Operator besitzt einen identischen Aufbau, nur daß hier zusätzliche Tuning-Parameter das XML-Retrieval ermöglichen. Im Gegensatz zum TextExtender können alle Tuning-Parameter des *etx_contains*-Operators auch im erweiterten Operator genutzt werden, da im Excalibur DataBlade die zutreffenden Terme lokalisiert werden können. Durch den Bezug der Tuning-Parameter zum Suchkriterium können Attribut/Wert-Anfragen und Typoperationen nicht über zusätzliche Tuning-Parameter realisiert werden. Um diese Anfragen ausführen zu können, wird zum einen eine Funktion *extract* konzipiert.

Voraussetzung:	column $\in$ {CLOB, Char, Varchar, LVarchar, IfxDocDesc}
Typ:	extract: column, row(string, string), SLV $\rightarrow$ list<string>

Diese Funktion unterscheidet sich vom *contains*-Operator nur durch ihren Rückgabetyp *list<string>*. Auch die Semantik der Tuning-Parameter ist identisch zum *contains*-Operator. Zum anderen werden Funktionen definiert, die einen Vergleich der Listenelemente mit einem als Argument übergebenen Wert (*value*) realisieren.

Voraussetzung:	$\tau \in$ {int, real, string, date}
	compare $\in$ {eq, le, lt, ge, gt, neq}
Typ:	compare: list<string>, $\tau \rightarrow$ bool
Syntax:	compare(extract, value)

Das Listenelement wird dazu in den Datentyp des Vergleichswertes konvertiert.

In den nachfolgenden Paragraphen werden Tuning-Parameter vorgestellt, die in Kombination mit den Tuning-Parametern des Excalibur DataBlades ein XML-spezifisches Retrieval ermöglichen. Die Vorstellung der Syntax und Semantik der konzipierten Tuning-Parameter erfolgt wie beim TextExtender durch eine Trennung in die Bereiche *einfache Strukturanfragen, Strukturanfragen mit Pfadausdrücken, Abstandsanfragen, Inklusionsanfragen, Typisierung, Strukturextraktion* und *Rangordnung.*

Einfache Strukturanfragen

1. XML-CONSTRUCT = ELEMENT

   Durch diesen Tuning-Parameter wird das im Suchkriterium angegebene Element im Dokument gesucht.

   Beispiel: *Gibt es Dokumente, die das Element Kapitel enthalten ?*

   ```
   SELECT docId
   FROM   dokument
   WHERE  contains(volltext,
                   row('Kapitel','XML-CONSTRUCT=ELEMENT'))
   ```

2. XML-CONSTRUCT = ELEMENT
   PATHEXPRESSION = *elementname*

   Durch die Kombination dieser Tuning-Parameter wird geprüft, ob das im Suchkriterium angegebene Element im Element *elementname* enthalten ist.

3. XML-CONSTRUCT = CONTENT

   Durch diesen Tuning-Parameter wird geprüft, ob die im Suchkriterium angegebenen Terme als Inhalt von Strukturelementen auftreten.

4. XML-CONSTRUCT = CONTENT
   PATHEXPRESSION = *elementname*

   Durch die Kombination dieser Tuning-Parameter wird geprüft, ob die im Suchkriterium angegebenen Terme im Inhalt des Strukturelementes *elementname* auftreten.

   Beispiel: *Gibt es Dokumente, die den Begriff Datenbanken im Titel enthalten ?*

```
SELECT docId
FROM   dokument
WHERE  contains(volltext, row('Datenbanken',
       'XML-CONSTRUCT=CONTENT & PATHEXPRESSION=Titel'))
```

5. XML-CONSTRUCT = ATTRIBUTE

   Durch diesen Tuning-Parameter wird geprüft, ob der im Suchkriterium angegebene Attributname im Dokument enthalten ist.

   Wird dieser Tuning-Parameter in der *extract*-Funktion verwendet, werden alle Werte des im Suchkriterium angegebenen Attributnamens durch die Funktion ermittelt und zurückgeliefert. Mit Hilfe der Vergleichsfunktionen können diese Attributwerte dann mit einem vorgegebenen Wert verglichen werden.

6. XML-CONSTRUCT = ATTRIBUTE
   PATHEXPRESSION = *elementname*

   Durch die Kombination dieser Tuning-Parameter wird geprüft, ob der im Suchkriterium angegebene Attributname im Element *elementname* enthalten ist.

   Werden diese Tuning-Parameter in der *extract*-Funktion verwendet, wird der Wert des im Suchkriterium angegebenen Attributnamen im Element *elementname* ermittelt und durch die Funktion zurückgegeben. Mit Hilfe der Vergleichsfunktionen kann dieser Attributwert dann mit einem vorgegebenen Wert verglichen werden.

   Beispiel: *Gibt es Dokumente mit dem Element Buch, das ein Attribut/ Wert-Paar Sprache=de enthält ?*

```
SELECT docId
FROM   dokument
WHERE  eq(extract(volltext, row('Sprache',
           'XML-CONSTRUCT=ATTRIBUTE & PATHEXPRESSION=Buch')),
           'de')
```

Strukturanfragen mit Pfadausdrücken

1. XML-CONSTRUCT = ELEMENT
   PATHEXPRESSION = *pathexpression*

   Durch die Kombination dieser Tuning-Parameter wird geprüft, ob das im Suchkriterium angegebene Element in dem durch *pathexpression* beschriebenen XML-Bereich enthalten ist.

2. XML-CONSTRUCT = CONTENT
   PATHEXPRESSION = *pathexpression*

   Durch die Kombination dieser Tuning-Parameter wird geprüft, ob die im Suchkriterium angegebenen Terme im Inhalt des durch *pathexpression* beschriebenen XML-Bereichs enthalten sind.

3. XML-CONSTRUCT = ATTRIBUTE
   PATHEXPRESSION = *pathexpression*

   Durch die Kombination dieser Tuning-Parameter wird geprüft, ob die Elemente des durch den Pfadausdruck *pathexpression* beschriebenen XML-Bereichs das im Suchkriterium angegebene Attribut enthalten.

   Durch Verwendung dieser Tuning-Parameter in der *extract*-Funktion können die Werte des im Suchkriterium angegebenen Attributnamens in den betroffenen Elementen des Pfadausdrucks *pathexpression* ermittelt und durch die Funktion zurückgegeben werden. Mit Hilfe der Vergleichsfunktionen können diese Attributwerte dann mit einem vorgegebenen Wert verglichen werden.

Abstandsfunktionen

1. XML-CONSTRUCT = CONTENT
   SEARCH_TYPE = PROX_SEARCH(k)
   $k \mid \{k \in \text{nat.Zahlen} \wedge k \geq 1\}$

   Durch die Kombination dieser Tuning-Parameter wird verifiziert, ob die Terme des Suchkriteriums gemeinsam im Inhalt eines beliebigen Strukturelements auftreten und dabei eine maximale Distanz k wahren.

2. XML-CONSTRUCT = CONTENT
   SEARCH_TYPE = PROX_SEARCH(k)
   PATHEXPRESSION = *elementname*
   $k \mid \{k \in \text{nat.Zahlen} \wedge k \geq 1\}$

   Durch die Kombination dieser Tuning-Parameter wird verifiziert, ob die Terme des Suchkriteriums gemeinsam im Inhalt des Elements *elementname* auftreten und zwischen den Termen eine maximale Distanz k besteht.

3. XML-CONSTRUCT = CONTENT
   SEARCH_TYPE = PROX_SEARCH(k)
   PATHEXPRESSION = *pathexpression*
   $k \mid \{k \in \text{nat.Zahlen} \wedge k \geq 1\}$

   Durch die Kombination dieser Tuning-Parameter wird verifiziert, ob die Terme des Suchkriteriums gemeinsam in dem durch *pathexpression* definierten Unterbaum auftreten und zwischen den Termen eine maximale Distanz k besteht.

Beispiel: *Welche Bücher hat der Autor Andreas Heuer geschrieben ?*

```
SELECT Titel.Obertitel
FROM   Buch_A
WHERE  contains(volltext, row('Andreas Heuer',
       'XML-CONSTRUCT=CONTENT & SEARCH_TYPE=PROX_SEARCH(5)
        & PATHEXPRESSION=Buch.Autor.* '))
```

4. XML-CONSTRUCT = ELEMENT
   DIRECTION = *direction*(k) OF *elementname*
   $k \mid \{k \in \text{nat.Zahlen} \wedge k \geq 1\}$
   $direction \in \{\text{RIGHT, LEFT, TOP, DOWN}\}$

   Durch die Kombination dieser Tuning-Parameter wird verifiziert, ob das Element des Suchkriteriums k Elemente vom Element *elementname* in Richtung *direction* entfernt ist.

   Beispiel: *Folgt zwei Strukturelemente nach der ISBN (auf gleicher Hierarchieebene) ein Volume ?*

```
SELECT DocId
FROM   Dokument
WHERE  contains(volltext, row('Volume',
       'XML-CONSTRUCT=ELEMENT & DIRECTION=RIGHT(2) OF ISBN'))
```

5. XML-CONSTRUCT = ELEMENT
   PATHEXPRESSION = *elementname1*
   DIRECTION = *direction*(k) OF *elementname2*
   $k \mid \{k \in \text{nat.Zahlen} \wedge k \geq 1\}$
   $direction \in \{\text{RIGHT, LEFT, TOP, DOWN}\}$

   Durch die Kombination dieser Tuning-Parameter wird verifiziert, ob das Element des Suchkriteriums ein direkter Nachfolger des Elementes *elementname1* ist und sich k Elemente vom Element *elementname2* in Richtung *direction* befindet.

6. XML-CONSTRUCT = ELEMENT
   PATHEXPRESSION = *pathexpression*
   DIRECTION = *direction*(k) OF *elementname*
   $k \mid \{k \in \text{nat.Zahlen} \wedge k \geq 1\}$
   $direction \in \{\text{RIGHT, LEFT, TOP, DOWN}\}$

   Durch die Kombination dieser Tuning-Parameter wird verifiziert, ob das Element des Suchkriteriums k Elemente vom Element *elementname* in Richtung *direction* entfernt ist. Das Element des Suchkriteriums muß dabei unterhalb des Pfadausdrucks *pathexpression* auftreten.

7. XML-CONSTRUCT = CONTENT
   DIRECTION = *direction*(k) OF *elementname*
   $k \mid \{k \in \text{nat.Zahlen} \wedge k \geq 1\}$
   $direction \in \{\text{RIGHT, LEFT, TOP, DOWN}\}$

   Durch die Kombination dieser Tuning-Parameter wird verifiziert, ob der Term des Suchkriteriums im Inhalt eines Elementes existiert. Trifft dies zu, muß dieses Element k Elemente vom Element *elementname* in Richtung *direction* entfernt liegen.

8. XML-CONSTRUCT = CONTENT
   PATHEXPRESSION = *elementname1*
   DIRECTION = *direction*(k) OF *elementname2*
   $k \mid \{k \in \text{nat.Zahlen} \wedge k \geq 1\}$
   $direction \in \{\text{RIGHT, LEFT, TOP, DOWN}\}$

   Durch die Kombination dieser Tuning-Parameter wird verifiziert, ob der Term des Suchkriteriums im Inhalt des Elementes *elementname1* existiert. Trifft dies zu, muß dieses Element k Elemente vom Element *elementname2* in Richtung *direction* entfernt liegen.

9. XML-CONSTRUCT = CONTENT
   PATHEXPRESSION = *pathexpression*
   DIRECTION = *direction*(k) OF *elementname*
   $k \mid \{k \in \text{nat.Zahlen} \wedge k \geq 1\}$
   $direction \in \{\text{RIGHT, LEFT, TOP, DOWN}\}$

   Durch die Kombination dieser Tuning-Parameter wird verifiziert, ob der Term des Suchkriteriums innerhalb des Pfadausdrucks *pathexpression* auftritt. Trifft dies zu, muß das Element in dem sich der Term befindet k Elemente vom Element *elementname* in Richtung *direction* entfernt liegen.

Inklusionsoperationen

1. XML-CONSTRUCT = CONTENT
   CARDINALITY = k
   $k \mid \{k \in \text{nat.Zahlen} \wedge k \geq 1\}$

   Durch die Kombination dieser Tuning-Parameter wird geprüft, ob der Term des Suchkriteriums mindestens k-mal in einem beliebigen Element auftritt.

2. XML-CONSTRUCT = CONTENT
   CARDINALITY = k
   PATHEXPRESSION = *elementname*
   $k \mid \{k \in \text{nat.Zahlen} \wedge k \geq 1\}$

   Durch die Kombination dieser Tuning-Parameter wird geprüft, ob der Term des Suchkriteriums mindestens k-mal im Element *elementname* auftritt.

3. XML-CONSTRUCT = CONTENT
   CARDINALITY = k
   PATHEXPRESSION = *pathexpression*
   $k \mid \{k \in \text{nat.Zahlen} \wedge k \geq 1\}$

   Durch die Kombination dieser Tuning-Parameter wird geprüft, ob der Term des Suchkriteriums mindestens k-mal im Inhalt des durch *pathexpression* beschriebenen XML-Bereichs enthalten ist.

4. XML-CONSTRUCT = ELEMENT
   CARDINALITY = k
   $k \mid \{k \in \text{nat.Zahlen} \wedge k \geq 1\}$

   Durch die Kombination dieser Tuning-Parameter wird geprüft, ob das Element des Suchkriteriums mindestens k-mal im Dokument enthalten ist.

5. XML-CONSTRUCT = ELEMENT
   CARDINALITY = k
   PATHEXPRESSION = *elementname*
   $k \mid \{k \in \text{nat.Zahlen} \wedge k \geq 1\}$

   Durch die Kombination dieser Tuning-Parameter wird geprüft, ob das Element des Suchkriteriums mindestens k-mal im Element *elementname* enthalten ist.

6. XML-CONSTRUCT = ELEMENT
   CARDINALITY = k
   PATHEXPRESSION = *pathexpression*
   $k \mid \{k \in \text{nat.Zahlen} \wedge k \geq 1\}$

   Durch die Kombination dieser Tuning-Parameter wird geprüft, ob das Element des Suchkriteriums mindestens k-mal in dem durch *pathexpression* beschriebenen XML-Bereich enthalten ist.

Typisierung

Wie zu Beginn dieses Abschnitts bereits eingeführt, werden für Typ- und Attribut/Wert-Anfragen keine zusätzlichen Tuning-Parameter konzipiert. Eine Typ- oder Attribut/Wert-Anfrage wird über die *extract*- und Vergleichsfunktionen formuliert.

```
compare(extract(column,row(Suchkriterium,Tuning-Parameter)),value)
```

Die Tuning-Parameter besitzen in der *extract*-Funktion die gleiche Semantik wie im *contains*-Operator. Im Unterschied zum *contains*-Operator kann bei der Selektion von Inhalten das Suchkriterium einen leeren String enthalten. In diesem Fall wird entweder der gesamte Inhalt extrahiert (keine Pfadinformation) oder nur der Inhalt des optionalen Pfadausdrucks. Der Vorteil der Herauslösung der Typanfragen ist, daß auch Vergleiche mit strukturierten Attributwerten dargestellt werden können. Im Gegensatz dazu können bei der TextExtender-Erweiterung nur Konstantenvergleiche realisiert werden. Die fehlenden Mengenkonstruktoren im DB2-Datenmodell würden diese Konzeption für die TextExtender-Erweiterung nicht zulassen.

1. XML-CONSTRUCT = CONTENT

   Durch diesen Tuning-Parameter und der Angabe eines leeren Suchkriteriums werden zunächst alle Inhalte in Form von Elementen einer Liste selektiert. Anschließend werden alle Elemente dieser Liste mit dem Wert — angegeben als 2. Parameter der Funktion — der verwendeten Vergleichsfunktion verglichen. Das Ergebnis ist *wahr*, wenn mindestens ein Element den Vergleich erfüllt.

2. XML-CONSTRUCT = CONTENT
   PATHEXPRESSION = *elementname*

   Durch die Kombination dieser Tuning-Parameter und der Angabe eines leeren Suchkriteriums wird der Inhalt des Elementes *elementname* extrahiert. Anschließend wird das Element dieser Liste (es können aber auch mehrere sein) mit dem Wert — angegeben als 2. Parameter der Funktion — der verwendeten Vergleichsfunktion verglichen.

3. XML-CONSTRUCT = CONTENT
   PATHEXPRESSION = *pathexpression*

   Durch die Kombination dieser Tuning-Parameter und der Angabe eines leeren Suchkriteriums wird der Inhalt aller Elemente unterhalb des Pfadausdrucks *pathexpression* extrahiert. Anschließend werden die Elemente dieser Liste mit dem Wert — angegeben als 2. Parameter der Funktion — der verwendeten Vergleichsfunktion verglichen. Das Ergebnis ist *wahr*, wenn mindestens ein Element den Vergleich erfüllt.

   Beispiel: *Gibt es Diplomarbeiten aus dem Jahr 1999?*

   ```
   SELECT DocId
   FROM Diplomarbeit
   WHERE eq(extract(volltext,row('','XML-CONSTRUCT=CONTENT &
                     PATHEXPRESSION=Diplomarbeit.Meta.Jahr ')
                     ), 1999 )
   ```

4. XML-CONSTRUCT = ATTRIBUTE

   Durch diesen Tuning-Parameter werden alle Werte des im Suchkriterium angegebenen Attributnamens extrahiert. Anschließend werden die Elemente dieser Liste mit dem Wert — angegeben als 2. Parameter der Funktion — der verwendeten Vergleichsfunktion verglichen. Das Ergebnis ist *wahr*, wenn mindestens ein Element den Vergleich erfüllt.

5. XML-CONSTRUCT = ATTRIBUTE
   PATHEXPRESSION = *elementname*

   Durch die Kombination dieser Tuning-Parameter wird der Wert des im Suchkriterium angegebenen Attributnamens im Element *elementname* extrahiert. Anschließend wird das Element dieser Liste (es können aber auch mehrere sein) mit dem Wert — angegeben als 2. Parameter der Funktion — der verwendeten Vergleichsfunktion verglichen.

6. XML-CONSTRUCT = ATTRIBUTE
   PATHEXPRESSION = *pathexpression*

   Durch die Kombination dieser Tuning-Parameter werden alle Werte des im Suchkriterium angegebenen Attributnamens in den betroffenen Elementen des Pfadausdrucks *pathexpression* extrahiert. Anschließend werden die Elemente dieser Liste mit dem Wert — angegeben als 2. Parameter der Funktion — der verwendeten Vergleichsfunktion verglichen. Das Ergebnis ist *wahr*, wenn mindestens ein Element den Vergleich erfüllt.

Strukturextraktion

Die Extraktion von Strukturbereichen aus Dokumenten erfolgt über die bereits beschriebene Funktion *extract*. Je nach Belegung des Tuning-Parameters XML-CONSTRUCT werden entweder Inhalte (CONTENT), Attributwerte (ATTRIBUTE) oder komplette Strukturbereiche (ELEMENT) ausgewählt. Da u.U. durch eine Anfrage mehrere zutreffende Bereich ausgewählt werden, erfolgt die Rückgabe des Ergebnisses in Form einer Liste.

Rangordnung

Auf die Rangordnung der Dokumente einer Anfrage kann beim Excalibur DataBlade über die *statement local variable* (SLV) projiziert werden. Angesichts identischer Schnittstellen wird für den erweiterten *contains*-Operator das gleiche Prinzip angewendet. Analog zum TextExtender müssen zur Berechnung der Rangordnung beim XML-Retrieval neue Kriterien berücksichtigt werden. Diese Kriterien entsprechen im wesentlichen den Rangordnungskriterien des TextExtenders aus Abschnitt 5.2.3.1.

5.3.2.3 Erfüllung der Kriterien nach der Konzeption

Abschließend wird nun validiert, inwieweit durch die Konzeption die zu Beginn dieses Kapitels (Abschnitt 5.1) aufgestellten Anforderungskriterien abgedeckt werden. Da insbesondere in den Änderungsoperationen die Gemeinsamkeiten zum TextExtender überwiegen, werden hier nur die Kriterien aufgeführt, die Unterschiede zum TextExtender aufweisen.

Kriterien für Änderungsoperationen

zu 3. *Indexaktualisierung nach Update:* Die Konzeption geht von der Verwendung des Excalibur-Index aus. Da die Aktualisierung durch ein Datenbankereignis gesteuert wird, kann davon ausgegangen werden, daß der Datenbankzustand der XML-Dokumente mit dem Zustand des Index harmoniert.

zu 6. *Indexaktualisierung nach Insert:* Siehe Punkt 3.

zu 7. *Indexaktualisierung nach Delete:* Siehe Punkt 3.

Kriterien für Anfrageoperationen

zu 1. *Trennung Struktur/Inhalt:* Die Realisierung der Trennung zwischen Termen, die als Strukturelemente interpretiert werden sollen und Termen, die einem Inhalt eines Elementes entsprechen, erfolgte in der Konzeption durch die Einführung des Tuning-Parameters XML-CONSTRUCT. Je nach Belegung dieses Parameters bezieht sich die Anfrage entweder auf den Inhalt eines Elementes (CONTENT) oder auf Strukturelemente (ELEMENT).

zu 2. *Attributanfragen:* Der Zugriff auf Attribute von Strukturelementen wird in der Konzeption durch den Tuning-Parameter XML-CONSTRUCT mit dem Wert ATTRIBUTE berücksichtigt. Zusätzlich kann der Attributwert mit Konstanten oder Werten der Datenbankattribute verglichen werden.

zu 3. *Bereichsauswahl:* Der Tuning-Parameter XML-CONSTRUCT kennzeichnet zunächst nur das XML-Strukturkonstrukt, das in der Suche betrachtet werden soll. Dabei wird jedoch völlig außer Acht gelassen, daß das semistrukturierte Dokument desweiteren eine logische Struktur besitzt. Diese logische Struktur wird in der Konzeption durch den Tuning-Parameter PATHEXPRESSION berücksichtigt, der die Spezifikation von Pfadausdrücken erlaubt. Diese Pfadausdrücke gewährleisten die Einschränkung des Terms auf einen speziellen Bereich der logischen Struktur.

zu 4. *Abstandsanfragen:* In der Konzeption wurden zwei Arten von Abstandsanfragen berücksichtigt. Zum einen können Anfragen formuliert werden, die das gemeinsame Auftreten von Termen im Inhalt von Strukturelementen prüfen. Diese Abstandsanfrage entspricht den Abstandsanfragen von Excalibur, wobei hier die Besonderheiten von XML beachtet wurden. Zum anderen wurden in die Konzeption Abstandsanfragen von Elementen aufgenommen. Diese Abstandsanfragen beinhalten einen Tuning-Parameter DIRECTION, der die Richtung anzeigt, in der das entfernte Elemente zu suchen ist.

zu 5. *Inklusionsbeziehungen:* Die Konzeption der Inklusionsbeziehungen berücksichtigt zum einen Anfragen nach der Vorkommenshäufigkeit von Termen in einem Strukturelement oder XML-Teilbaum über den Tuning-Parameter

CARDINALITY. Zum anderen werden Anfragen unterstützt, die eine Überprüfung des Enthaltenseins von Elementen in anderen Elementen oder Teilbäumen vorsehen. In diesen Anfragen können ebenfalls Vorkommenshäufigkeiten spezifiziert werden.

zu 6. *Typisierung:* Die aktuelle XML-Spezifikation enthält keine Festlegungen über konkrete Datentypen der Strukturelemente. Um aber eine ähnliche Mächtigkeit der Vergleichsanfragen wie auf den strukturierten Attributen des Datenbankschemas zu erreichen, wurde die Möglichkeit der Typanfragen in die Erweiterung aufgenommen. Für Typanfragen wurden keine zusätzlichen Tuning-Parameter konzipiert. Diese Anfragen können durch Extraktion der Inhalte und anschließenden Vergleich realisiert werden.

zu 7. *Indizierung:* Die Konzeption beinhaltet die Erweiterung der inhaltsbasierten Suche auf die Berücksichtigung der impliziten XML-Struktur. Die Erweiterung umfaßt jedoch keine zusätzliche Indizierung der Strukturelemente. Vielmehr wird der durch Excalibur erzeugt Index verwendet, um eine Vorselektion der Dokumente zu erreichen. Um auszuschließen, daß dabei bereits Dokumente herausgefiltert werden, deren Strukturelemente Bezeichnungen von Stoppworten tragen, kann bei der Indizierung der Dokumente der Indexparmeter INCLUDE_STOPWORDS=TRUE angegeben werden. Dadurch werden die Begriffe der Stoppwortliste zwar indiziert; sie werden jedoch nur berücksichtigt, falls im Suchausdruck der Tuning-Parameter INCLUDE_STOPWORDS angegeben wird.

5.3.2.4 Zusammenfassung

Die Beurteilung des Excalibur DataBlades hinsichtlich der Anforderungskriterien aus Abschnitt 5.1 ergab, daß das DataBlade gleichermaßen wie der TextExtender nicht für das Information-Retrieval semistrukturierter Dokumente geeignet ist. Auch die Abbildung der XML-Strukturen auf das objektrelationale Datenmodell von Informix wirft die in 5.2.2 genannten Probleme auf. Um auf Unterschiede zwischen DB2 und Informix hinzudeuten, wurde die Konzeption der TextExtender-Erweiterung auf Excalibur adaptiert. Auf die Abweichungen zwischen beiden Konzeptionen wird in Abschnitt 5.4 eingegangen. Die für die Erweiterung zusätzlich konzipierten Tuning-Parameter berücksichtigen die implizite Struktur der XML-Dokumente, was durch die abschließende Validierung der Anforderungskriterien gezeigt wurde.

5.4 Fazit

Zum Abschluß dieses Kapitels wird kurz auf die signifikantesten Unterschiede der Konzeptionen für DB2 und Informix eingegangen. Differenzen zwischen den

Konzeptionen zeigten sich u.a. :

- in der Abbildung des EER-Modells auf das entsprechende Datenmodell,

  Das Informix-Datenmodell unterstützt mehr objektorientierte Konzepte, so daß eine direkte Transformation des EER-Modells in das Informix-Datenmodell erfolgen kann. Bei der Abbildung des EER-Modells auf das DB2-Datenmodell müssen hingegen erst die mengenwertigen Attribute auf getrennte Relationen normalisiert werden.

- in den Möglichkeiten zur Nutzung der Funktionalität der Retrieval-Module in den Erweiterungen,

  Die Excalibur Tuning-Parameter des *etx_ contains*-Operators können vollständig in der Erweiterung zum Einsatz kommen, da im DataBlade durch die Highlight-Funktion auf die zutreffenden Terme einer Anfrage geschlossen werden kann. Die Erweiterung des TextExtenders kann indessen den überwiegenden Teil der linguistischen Funktionalität des TextExtenders nicht nutzen. Dennoch verfügt die auf Excalibur basierende Erweiterung nicht über wesentlich mehr Funktionalität, da das DataBlade von vornherein nicht die Leistungsfähigkeit des TextExtenders besitzt.

- und in der Anfragegestaltung.

  Eine Anfrage an die TextExtender-Erweiterung kann eine beliebige Kombination verschiedener Strukturkonstrukte enthalten. Im Unterschied dazu wird in Excalibur durch den Bezug aller Terme des Suchkriteriums auf die gleichen Tuning-Parameter verhindert, daß Anfragen, die Kombinationen verschiedener Strukturkonstrukte beinhalten, durch einen Operator-Aufruf der Excalibur-Erweiterung formuliert werden können. Die Anfrage

  Hat Beate Porst eine Diplomarbeit geschrieben, und enthält diese Diplomarbeit ein Vorwort?

  kann daher nicht durch einen einzelnen *contains*-Aufruf realisiert werden. Im Falle, daß in Informix UDFs in der WHERE-Klausel generell nicht verknüpft werden können (wie in Excalibur), muß diese Anfrage mittels geschachtelter SFW-Blöcke umgesetzt werden.

Die Konstrukte der Konzeptionen bilden zunächst nur eine Basis um Inhalts- und Strukturanfragen auf semistrukturierten Dokumenten zu ermöglichen. Weitere Funktionen, wie objketerhaltende und objekterzeugende Transformationen, Restrukturierung, Aggregation und Mengenoperationen werden im nächsten Kapitel als Ausblick auf einen "idealen" XML-Datentypen vorgestellt.

Kapitel 6

Konzeption eines allgemeinen XML-Datentyps

Das vorangegangene Kapitel hat gezeigt, daß grundsätzlich Möglichkeiten bestehen, ein im Datenbanksystem integriertes Volltext-Retrieval-Modul auf ein XML-spezifisches Retrieval zu erweitern. Zunächst wurde der Vorschlag unterbreitet, die DTD-Strukturen auf ein objektrelationales Datenbankschema abzubilden und auf den Inhalten der so entstandenen Attribute das existierende Volltext-Retrieval anzuwenden. Dieser Ansatz wurde dennoch nicht weiter verfolgt. Einerseits, weil die Abbildung einer DTD auf ein Datenbankschema in der Regel nicht trivial ist und andererseits aufgrund der in Abschnitt 5.2.2 beschriebenen Probleme. Statt dessen wurde die Anfragesyntax der Volltext-Retrieval-Systeme auf das XML-Retrieval erweitert, wodurch als Volltext gespeicherte semistrukturierte Dokumente angefragt werden können. Die Probleme dieses Vorschlages sind einerseits der fehlende Strukturindex, wodurch nur eine sequentielle Suche realisiert werden kann. Dies führt insbesondere bei sehr großen Dokumenten zu einem inakzeptablen Zeitverhalten. Andererseits ist die dargestellte XML-Erweiterung stark von den Gegebenheiten des zugrunde liegenden Volltext-Retrieval-Systems abhängig, so daß beispielsweise beim TextExtender die linguistische Funktionalität nicht in das XML-Retrieval-System übernommen werden konnte. Für ein "ideales" XML-System ebenfalls nicht akzeptabel ist die Orientierung der XML-Syntax an der Syntax des Volltext-Retrieval-Systems, da hier keine struktur- und hierarchieunterstützenden Konzepte existieren.

In diesem Kapitel wird ein Vorschlag eines XML-Datentyps sowie Funktionen, die auf diesem Datentyp operieren unterbreitet, wodurch Retrieval-Anfragen auf XML-Dokumenten innerhalb eines objektrelationalen Datenbanksystems realisiert werden können. Bei der Vorstellung der Funktionen wird von konkreten Systemen oder Realisierungsmöglichkeiten abstrahiert, so daß in gewissem Sinne von einem "idealen" XML-Datentyp gesprochen werden kann.

6.1 Existierende Anfragesprachen

Die Aktualität der Thematiken *XML* und *Anfragesprachen für XML* zeigt sich deutlich anhand der Veröffentlichungen im Bearbeitungszeitraum dieser Arbeit. Bezüglich Anfragesprachen lagen zu Beginn nur sehr wenige speziell auf XML ausgerichtete Veröffentlichungen vor. Inzwischen existieren eine ganze Reihe konkreter Vorschläge. Drei Vorschläge für Anfragesprachen auf XML-Dokumenten sollen hier kurz vorgestellt werden. Herausgestellt werden dabei Aspekte, die in den Sprachen bisher nicht berücksichtigt wurden, um so die Notwendigkeit eines weiteren Vorschlages zu begründen.

6.1.1 XML-QL

XML-QL [DFF+98] war der erste Vorschlag einer Anfragesprache für XML-Dokumente. Die Fokussierung dieser Anfragesprache liegt auf EDI-Anwendungen[1], mit deren Hilfe XML-Dokumente bzw. Teile solcher Dokumente ausgetauscht werden können. In diesem Vorschlag werden XML-Dokumente aus Datenbanksicht betrachtet, das heißt ein XML-Dokument entspricht einer Datenbank und eine DTD einem Datenbankschema. Mit Hilfe von XML-QL-Ausdrücken können Anfragen formuliert werden, die Teile eines Dokumentes extrahieren oder transformieren.

Eine einfache XML-QL-Anfrage hat die Form

WHERE	*condition*
CONSTRUCT	*output*

In der Bedingung können Elementmuster spezifiziert werden, die zur Selektion in der angegebenen Datenquelle enthalten sein müssen. Die Spezifikation der Elementmuster kann Variablen enthalten, die an die entsprechenden Elemente oder Inhalte der Datenquelle gebunden werden. Diese im WHERE-Block definierten Variablen dürfen dann in dieser oder in den eingeschlossenen WHERE-CONSTRUCT-Anweisungen auf der rechten Seite eines Prädikates verwendet werden. Durch die Nutzung von Variablen in der Anfragesprache kann auf Strukturbereiche oder Inhalte, die an diese Variablen gebunden wurden, in anderen Teilen der Anfrage Bezug genommen werden. Dadurch können beispielsweise Joins realisiert werden, indem Elemente verglichen werden, die den gleichen Wert enthalten müssen. Durch die Nestung der WHERE-CONSTRUCT-Anweisung innerhalb des CONSTRUCT-Blockes wird u.a. die Gruppierung des Ergebnisses ermöglicht. Neben den bereits genannten Operationen kann in XML-QL eine

[1] EDI - Electronic Data Interchange

Ordnung des Ergebnisses festgelegt und eine Integration von Daten verschiedener Quellen in ein Ergebnisdokument vorgenommen werden. Zur Spezifikation der Elementmuster können neben einfachen Begin- und End-Tags auch reguläre Pfadausdrücke verwendet werden. In der nachfolgenden Beispielanfrage wird eine Liste aller Autoren und Titel des Eingabedokumentes zurückgegeben, falls das Dokument ein Element *Buch* besitzt, das wiederum mindestens ein Element *Autor*, ein Element *Titel* und ein Element *Verlag* enthält. Zur Selektion eines Dokumentes muß der Name des Verlages den Wert *Addison-Wesley* enthalten.

```
WHERE <Buch>
          <Verlag><Verlagsname>Addison-Wesley</></>
          <Titel> $t </>
          <Autor> $a </>
      </> IN "www.xmlexample/buch.xml"
CONSTRUCT <result>
              <Autor> $a </>
              <Titel> $t </>
          </>
```

In der Anfrageformulierung kann in XML-QL für das End-Tag die verkürzte Schreibweise </> verwendet werden. Im WHERE-Block der vorangegangenen Anfrage wurden die Variablen $t und $a an die Inhalte der Elemente *Titel* und *Autor* gebunden. Falls der Bedingungsteil der Anfrage wahr ist, werden die Inhalte zur Konstruktion des Ergebnisses verwendet.

In der Anfragesprache werden zur Zeit Operationen wie Aggregation und das Ranking des Ergebnisses nicht unterstützt. Weiterhin können weder reguläre Ausdrücke noch linguistische Parameter für die im Bedingungsteil enthaltenen Terme angegeben werden. Mit Hilfe von XML-QL können XML-Dokumente angefragt werden. Es ist jedoch nicht möglich, das Dokument zu ändern, zu löschen oder ein neues Dokument einzufügen. Aus der Beschreibung der Anfragesprache ging desweiteren nicht klar hervor, wie mit unterschiedlichen Datentypen umgegangen wird. Es wurden zwar Beispielanfragen formuliert, die Vergleiche auf verschiedenen Datentypen durchführen; wie jedoch die Differenzierung der Datentypen vorgenommen wird, war nicht ersichtlich.

6.1.2 XML Query Language (XQL)

XQL [RLS98] ist eine Anfragesprache zur Adressierung und Selektion von Elementen und Text eines XML-Dokumentes. Der XQL-Vorschlag kann als Obermenge der Extensible Stylesheet Language (XSL[2]) betrachtet werden. XSL be-

[2] http://www.w3.org/TR/WD-xsl

steht aus einer Sprache zur Transformation von XML-Dokumenten und einem Vokabular zur Spezifikation von Formatierungssemantiken. In XSL können syntaktische Muster definiert werden, die Elemente innerhalb eines XML-Dokumentes identifizieren. Die XQL beinhaltet zusätzlich u.a. reichhaltigere Vergleichsoperationen, Datentypkonvertierungen, Indexierung einer Collection und Vereinigung und Durchschnitt von Elementmengen. Die in 6.1.1 dargestellte Beispielanfrage kann in XQL folgendermaßen formuliert werden:

```
<xsl:for-each select = “Buch[ Verlag/Verlagsname = “Addison-Wesley”
                                 $and$ Autor $and$ Titel ]” >
        <result>
           <Titel><xsl:value-of select=”Titel” /></Titel>
           <Autor><xsl:value-of select=”Autor” /></Autor>
        </result>
</xsl:for-each>
```

Obwohl die XQL für die Navigation innerhalb eines XML-Dokumentes umfangreiche Konstrukte bereitstellt, ist sie in den Anfragemöglichkeiten schwächer einzuschätzen als der XML-QL-Ansatz. In XQL beziehen sich Anfragen beispielsweise generell auf genau ein Dokument. Daraus folgt auch, daß eine Integration von Daten aus verschiedenen Datenquellen hier nicht realisierbar ist. Aufgrund des fehlenden Variablenkonzeptes können in XQL keine Joins formuliert werden. Weiterhin erstrecken sich die Aggregat-Operationen allein auf den *count*-Operator. Eingeschränkt wird die Anfragemöglichkeit auch dadurch, daß die zur Selektion von Elementen angegebenen Muster keine regulären Ausdrücke oder Zeichenmaskierungen enthalten dürfen. Entsprechend dem XML-QL-Ansatz sind Dokument-Manipulationsoperationen durch die Anfragesprache nicht realisiert.

6.1.3 XQuery

XQuery [DeR98] ist ein Vorschlag für eine Anfragesprache, die auf der Syntax der XPointer[3] aufbaut. XPointer beschreiben einen Ort oder Bereich innerhalb eines XML-Dokumentes, indem sie sich der Struktur der Dokumente bedienen. Der entscheidende Grund für die Nutzung der XPointer in XQuery waren die vorhandenen Operatoren für den navigierenden Zugriff. Einer der wichtigsten Begriffe in XQuery ist die *Location*. In XQuery wird der Begriff als abstrakter Datentyp aufgefaßt, der Dokumententeile oder Textpositionen beschreibt. In der Anfragesprache werden *Locations* als Mengen aufgefaßt. Jede XQuery-Anfrage besteht aus einer Folge von Selektionsschritten, die ähnlich wie OQL-Pfadausdrücke

[3] http://www.w3.org/TR/1998/WD-xptr-19980303

durch Punktnotation verbunden sind. Jeder Schritt selektiert Knoten des XML-Dokumentes entweder durch absolute Verweisausdrücke oder relative zu einer vorangegangenen Selektion. Bei der Ausführung eines Selektionsschrittes wird eine *Location set* auf eine neuen *Location set* abgebildet. Die initiale *Location set* einer Anfrage kann ein oder mehrere Dokumente umfassen. Enthält die initiale Menge mehrere Dokumente, so muß auf diesen Dokumenten eine Ordnung definiert sein, die beibehalten wird, solange im *Location set* Referenzen aus verschiedenen Dokumenten enthalten sind. Das Ergebnis einer XQuery-Anfrage ist wiederum eine *Location set*, da generell nur eine Transformation einer *Quell-Location set* in eine *Ziel-Location set* aufgrund einer Selektion vorgenommen wird. Grundsätzlich ist eine Nestung dieser Mengen nicht zulässig. *Location sets* können Referenzen auf beliebige Elemente, Attribute oder Selektionsbereiche enthalten.

Als Beispiel dient wiederum die Anfrage aus 6.1.1. Im Gegensatz zu den Sprachen XQL und XML-QL ist das Ergebnis der XQuery-Anfrage kein XML-Dokument, sondern eine *Location set*, die eine Menge von Referenzen auf Titel- und Autorknoten enthält.

```
union (
   intersection (
      (descendant (Buch &
                    (child(Verlag).child(Verlagsname = 'Addison-Wesley')))),
      (descendant(Buch & (child(Titel)))),
      (descendant(Buch & (child(Autor))))
                 ).child(Autor),
   intersection (
      (descendant (Buch &
                    (child(Verlag).child(Verlagsname = 'Addison-Wesley')))),
      (descendant(Buch & (child(Titel)))),
      (descendant(Buch & (child(Autor))))
                 ).child(Titel),
)
```

Die inneren *Intersection*-Operationen bilden zwei Mengen, die Referenzen auf Bücherknoten enthalten. Durch die *Union*-Operation werden aus der ersten Menge die Referenzen der direkten Autor-Nachfolger und aus der zweiten Menge die Referenzen der direkten Titel-Nachfolger selektiert und vereinigt.

Neben den aus XPointer bekannten Navigationskonstrukten wurden in XQuery Datentypen eingeführt, so daß Inhalts- und Attributwert-Vergleiche auf verschiedenen Datentypen ausgeführt werden können. Durch die Vergleichsmöglichkeiten und die Mengenbildung von XQuery können hier Joins realisiert werden, die sich aber nur auf ein Dokument beziehen.

Durch die Verwendung der *Location sets* sind in XQuery einige Operationen nicht realisiert. So werden keine Aggregations-Operationen, Ergebnis-Restrukturierungen, Transformationen oder Sortierungen unterstützt. Ähnlich wie in XQL beziehen sich Anfragen in der Regel auf ein Dokument. Bezieht sich eine Anfrage auf eine Menge von Dokumenten, so bestehen zwischen diesen Dokumenten Abhängigkeiten. Wie in XML-QL und XQL sind Dokument-Manipulationsoperationen durch die Anfragesprache nicht realisiert.

6.1.4 Fazit

Alle vorgestellten Sprachen dienen lediglich dem Zweck, durch Anfragen eine Selektion, Restrukturierung, Kombination, Aggregation oder Ableitung aus einem oder mehreren Dokumenten zu erzielen. Das Einfügen, Löschen oder Ändern von XML-Dokumenten wird hingegen von keiner Sprache unterstützt. Auch in den Anfragesprachen selbst fehlen gänzlich Operationen zur Bildung einer Rangordnung und zum Text-Retrieval. Die Tabelle 6.1 zeigt zusammenfassend, welche Funktionalität durch welche Sprache berücksichtigt wird.

	XML-QL	**XQL**	**XQuery**	**TextExt.**
Aggregation	-	$\mp$	-	-
Ranking	-	-	-	$\mp$
Insert	-	-	-	-
Update	-	-	-	-
Delete	-	-	-	-
Join	+	-	$\mp$	-
Restrukturierung	+	+	-	-
Transformation	+	+	-	+
Anfrage an eine Dokumentenmenge	+	-	$\mp$	+
Text-Retrieval	-	-	-	$\mp$
reguläre Ausdrücke in Elementnamen	+	-	+	+
Navigationsoperationen	+	+	+	+
Vergleichsoperationen	+	+	+	+
Mengenoperationen	+	+	+	-
Boolesche Operationen	+	+	+	+
Unterstützung verschiedener Datentypen	+	+	+	+

Tabelle 6.1: Unterstützte Funktionalität der XML-Anfragesprachen

Zum Vergleich wird durch die Tabelle 6.1 auch widergegeben, welche Funktionalität durch die in Abschnitt 5.2.3.2 konzipierten Konstrukte des TextExtenders unterstützt wird. Die Operationen des zu konzipierenden allgemeinen XML-Datentyps sollten dann idealerweise alle in der Tabelle aufgeführten Funktionen umfassen.

6.2 Der allgemeine Datentyp

Die Konzeption eines allgemeinen XML-Datentyps setzt sich aus einem spezifischen Datentyp und aus Funktionen, die auf diesem Datentyp operieren, zusammen. Zur Verwaltung und Anfrage der XML-Dokumente einer Datenbank wird der spezifische Datentyp *xml* erzeugt, dessen Basisdomäne vom Typ CLOB ist. Da sich der Datentyp ohne zusätzliche Operationen nicht von CLOB unterscheidet, werden nachfolgend typspezifische Operationen vorgestellt, die diesen Typ von CLOB abgrenzen.

6.2.1 Anfrageoperationen

Der Zweck einer XML-Anfrage ist u.a. die Selektion, Restrukturierung, Kombination, Ableitung und Aggregation von Dokumenten bzw. von Teilen von Dokumenten. Relevante Kriterien sind dabei die hierarchische und sequentielle Struktur, Attributwerte, Elemente und Elementinhalte, die Interpretation von Elementinhalten sowie die Bewertung der Güte eines Dokumentes in Bezug auf die gestellte Anfrage. Der in dieser Arbeit unterbreitete Vorschlag einer Anfragesprache basiert auf der *select-from-where*-Anweisung (SFW) der Anfragesprache SQL. Für die Verwendung sprachen zum einen die klare Semantik der Anweisung und zum anderen die Verwaltung der Dokumente durch ein Datenbanksystem, wodurch Kombinationen zwischen XML- und anderen Attributen des Datenbankschemas innerhalb einer Anfrage möglich sind. Ein einfacher SWF-Block hat die Form:

SELECT *projektion*
FROM *dokument*
WHERE *bedingung*

6.2.1.1 Pfadausdrücke

- Absolute Pfadausdrücke

  Durch einen absoluten Pfadausdruck werden ein oder mehrere XML-Strukturbereiche beschrieben. Startpunkt kann ein Dokument der FROM-Klausel oder eine Pfadvariable (siehe Abschnitt 6.2.1.10) sein. Durch anschließende Notation eines Pfeils (->) und der Angabe eines Elementes bzw. eines regulären Ausdrucks (*SimpleAPE*) kann in der XML-Baumstruktur abgestiegen werden. Als absoluter Pfadausdruck wird auch das Ergebnis einer Mengenoperation θ ($\theta \in \{+,-,/$) auf zwei *SimpleAPE* bezeichnet. Die nachfolgenden, formal nicht vollständigen EBNF-Regeln beschreiben die absoluten Pfadausdrücke.

```
AbsolutePathExpression ::= SimpleAPE
                         | ComplexAPE

SimpleAPE              ::= XmlArea ( ’->’ Elementname )*

ComplexAPE             ::= ’(’ SimpleAPE SetOP SimpleAPE ’)’

SetOP                  ::= ’+’
                         | ’-’
                         | ’/’

XmlArea                ::= Pfadvariable
                         | Dokument der FROM-Klausel

Elementname            ::= Konkreter Elementname
                         | Regulärer Ausdruck
```

- Relative Pfadausdrücke

  Die Lokalisierung eines relativen Pfadausdrucks erfolgt nur innerhalb des durch den absoluten Pfadausdruck festgelegten XML-Strukturbereichs. Innerhalb des relativen Pfadausdrucks können neben Elementnamen auch die Wiederholungsoperatoren + und * verwendet werden. Wird als Pfadausdruck lediglich ein Stern (*) spezifiziert, so wird der gesamte XML-Bereich des absoluten Pfadausdrucks als Basis verwendet. Besteht der relative Pfadausdruck lediglich aus einem Plus (+), ist die Basis das Wurzelelement des XML-Bereiches, der durch den absoluten Pfadausdruck beschrieben wird. Die nachfolgenden, formal nicht vollständigen EBNF-Regeln beschreiben die relativen Pfadausdrücke.

```
RelativePathExpression ::= Elementname ( ’->’ Elementname )*

Elementname            ::= Konkreter Elementname
                         | Regulärer Ausdruck
                         | ’*’
                         | ’+’
```

6.2.1.2 Selektion eines XML-Hierarchiebereiches

- IN

Voraussetzung:	a : absoluter Pfadausdruck
	f : boolesche Funktion
Typ:	f IN *xml*
Syntax:	f IN a

Dieser Operator qualifiziert einen XML-Bereich durch die Angabe eines absoluten Pfadausdrucks (Abschnitt 6.2.1.1), auf dem die Operation der booleschen Funktion f ausgeführt wird. Pfadangaben der Funktion f beziehen sich immer relativ auf den Kontext des qualifizierten XML-Bereiches.

Beispiel:

```
SELECT   Kapitel->* IN d->Buch->Body
FROM     dokument d
```

Durch die Anfrage erfolgt eine Projektion auf alle Kapitel des XML-Bereichs *d->Buch->Body*.

6.2.1.3 Navigationsoperationen

- first, last

Voraussetzung:	a : relativer Pfadausdruck
Typ:	first: *string* $\rightarrow$ *xml*
Syntax:	a.first()

Diese Funktion selektiert den *ersten* durch a gegebenen Strukturbereich, relativ zu dem in IN spezifizierten absoluten XML-Bereich.

Voraussetzung:	a : relativer Pfadausdruck
Typ:	last: *string* $\rightarrow$ *xml*
Syntax:	a.last()

Diese Funktion selektiert den *letzten* durch a gegebenen Strukturbereich, relativ zu dem in IN spezifizierten absoluten XML-Bereich.

Beispiel:

```
SELECT   Kapitel->*.first() IN d->Buch->Body
FROM     dokument d
```

Durch die Anfrage wird auf das erste Kapitel des XML-Bereiches *d->Buch->Body* projiziert.

- before, after

Voraussetzung:	$a,\ b$: relative Pfadausdrücke
	k : $\{k \in \text{nat.Zahlen} \wedge k \geq 1\}$ - Distanzparameter
Typ:	before: *string, string, int* $\rightarrow$ *xml*
Syntax:	a.before($b,\ k$)

Diese Funktion selektiert einen Strukturbereich, der durch den absoluten XML-Bereich in IN und den relativen Pfadausdruck a festgelegt wird, falls ein XML-Strukturbereich b existiert, der in der gleichen Hierarchietiefe maximal k-Knoten nach a auftritt.

Voraussetzung:	$a,\ b$: relative Pfadausdrücke
	k : $\{k \in \text{nat.Zahlen} \wedge k \geq 1\}$ - Distanzparameter
Typ:	after: *string, string, int* $\rightarrow$ *xml*
Syntax:	a.after($b,\ k$)

Diese Funktion selektiert einen Strukturbereich, der durch den absoluten XML-Bereich in IN und den relativen Pfadausdruck a festgelegt wird, falls ein XML-Strukturbereich b existiert, der in der gleichen Hierarchietiefe maximal k-Knoten vor a auftritt.

Beispiel:

```
SELECT   Titel->*.before(Verlag,4) IN d
FROM     dokument d
```

Durch die Anfrage wird auf Titel des XML-Bereiches d projiziert, in deren Nähe (maximale Distanz vier Knoten nach *Titel*) Verlage beschrieben werden.

6.2.1.4 Quantoren

- ALL

Voraussetzung	f : Prädikatfunktion
Syntax:	ALL f

Der All-Quantor iteriert über alle Strukturelemente entsprechend des Pfadausdrucks. Das Prädikat muß für alle zutreffenden Strukturelemente erfüllt sein.

- ANY

Voraussetzung	f : Prädikatfunktion
Syntax:	ANY f

Der ANY-Quantor iteriert solange über die zutreffenden Strukturbereiche, bis für einen Bereich das Prädikat erfüllt ist. Wird kein Quantor in der Anfrage angegeben, wird standardmäßig ANY angenommen.

Beispiel:

```
SELECT   d
FROM     dokument d
WHERE    ALL Kapitel->*.contains(“Datenbank”) IN d
```

Die Anfrage selektiert Dokumente, die in allen Kapiteln den Begriff “Datenbank” enthalten.

- exists

Voraussetzung:	a : relativer Pfadausdruck oder Navigationsfunktion
	$\tau \in \{string, xml\}$
Typ:	exists: $\tau \rightarrow bool$
Syntax:	a.exists()

Diese Funktion testet, ob der durch einen relativen Pfadausdruck oder durch eine Navigationsfunktion beschriebene XML-Strukturbereich im Kontext des zu untersuchenden Bereiches enthalten ist.

Beispiel:

```
SELECT d
FROM   dokument d
WHERE  Kapitel.exists() IN d
```

Die Anfrage selektiert ein Dokument, falls dieses mindestens ein Kapitel enthält.

6.2.1.5 Attributselektion

- attr

Voraussetzung:	a : relativer Pfadausdruck oder Navigationsfunktion
	b : Attributname
	$\tau \in \{string,\ xml\}$
Typ:	attr: τ, $string \rightarrow string$
Syntax:	a.attr(b)

Durch diese Funktion wird der Attributwert des Attributes b selektiert. Das Attribut muß sich dabei in dem XML-Strukturbereich befinden, der durch den absoluten XML-Bereich in IN sowie den relativen Pfadausdruck oder die Navigationsfunktion a beschrieben wird.

Beispiel:

```
SELECT d
FROM   dokument d
WHERE  Kapitel.attr(ident).exists() IN d
```

Die Anfrage selektiert ein Dokument, falls dieses Dokument ein Kapitel enthält und dieses Kapitel wiederum ein Attribut *ident* beinhaltet.

6.2.1.6 Aggregatoperationen

- count

Voraussetzung:	a : relativer Pfadausdruck
Typ:	count: $string \rightarrow int$
Syntax:	a.count()

Diese Funktion ermittelt die Anzahl relativer Strukturbereiche a innerhalb des absoluten Strukturbereichs. Als Aggregatoperation wurde nur die *count*-Operation definiert, da numerische Aggregatfunktionen durch Typkonvertierung realisiert werden können.

Beispiel:

```
SELECT d
FROM   dokument d
WHERE  Kapitel.count() > 5 IN d
```

Die Anfrage selektiert alle Dokumente, die mehr als 5 Kapitel enthalten.

6.2.1.7 Volltext-Retrieval

- contains

Voraussetzung:	a : relativer Pfadausdruck oder Navigationsfunktion
	b : Volltext-Retrieval-Ausdruck
	$\tau \in \{string,\ xml\}$
Typ:	contains: τ, $string \rightarrow bool$
Syntax:	a.contains(b)

Mit Hilfe dieser Funktion kann ein Volltext-Retrieval auf dem durch a beschriebenen XML-Strukturbereich realisiert werden.

Beispiel:

```
SELECT d
FROM   dokument d
WHERE  Kapitel->Titel->*.contains(STEMMED OF ‘‘Datenbank’’)
       IN d
```

Die Anfrage selektiert alle Dokumente, die in einer Kapitelüberschrift eine Stammform des Begriffes “Datenbank” enthalten.

6.2.1.8 Datentypkonvertierung

- value

Voraussetzung:	a : Attributwert
	b ∈{INT, REAL, DATE, STRING}
	$\tau \in \{int,\ real,\ date,\ string\}$
Typ:	value: $string,\ string \rightarrow \tau$
Syntax:	a.value(b)

Diese Funktion konvertiert den als Zeichenkette selektierten Attributwert a in den durch b vorgegebenen Datentyp.

Beispiel:

```
SELECT A->Kapitel->*
FROM   dokument d
WHERE  Kapitel.attr(ident).value(INT) = 6 IN d AS A
```

Durch die Anfrage werden im Strukturbereich d Kapitel lokalisiert, die ein Attribut *ident* enthalten. Der Attributwert dieses Attributes wird in den Datentyp *int* konvertiert und mit dem Wert 6 verglichen. Das Ergebnis der Anfrage ist eine Projektion auf die Kapitel, die diese Bedingung erfüllen.

- content

Voraussetzung:	a : relativer Pfadausdruck
	$b \in$ {INT, REAL, DATE, STRING}
	$\tau \in \{int,\ real,\ date,\ string\}$
Typ:	content: *string, string* $\rightarrow \tau$
Syntax:	a.content(b)

Diese Funktion konvertiert den Inhalt des XML-Strukturbereichs a in den durch b vorgegebenen Datentyp.

Beispiel:

```
SELECT d
FROM   dokument d
WHERE  Autor->Vorname.content(STRING) = ‘‘Andreas’’
       IN d->Buch->Meta
```

Die Anfrage selektiert alle Dokumente, die einen Autor enthalten, dessen Vorname “Andreas” lautet.

- convert

Voraussetzung:	$\tau \in \{int,\ real,\ date,\ string\}$
	a : Attribut vom Datentyp τ
	b : Elementname, der den konvertierten Wert umschließt
Typ:	convert: *string, string* $\rightarrow$ *xml*
Syntax:	a.convert(b)

Diese Funktion konvertiert einen Attributwert der Datenbank vom Typ τ in den Datentyp *xml*. Dieser Wert repräsentiert den Inhalt des durch b festgelegten Elementes.

Beispiel:

```
SELECT F.Preis.convert(REAL, Preis)
FROM   Flug F
```

Die Anfrage selektiert alle Preise aus der Flug-Relation und konvertiert diese Preise vom Typ *real* nach *xml*. Ein Ergebniswert könnte dann folgendermaßen aussehen:

```
<Preis>1049.00</Preis>
```

6.2.1.9 Gruppierung

- group

Voraussetzung:	$\vartheta \in \{AND,\ OR,\ NOT\}$
	$f_1, ..., f_n$: Prädikatfunktionen
Typ:	group: $bool,...,bool \rightarrow bool$
Syntax:	$(f_1 \vartheta ... \vartheta f_n)$

Durch die Gruppierung können mehrere Funktionen zusammengefaßt (Verknüpfung durch logische Operatoren) werden, deren Auswertung sich stets auf den selben absoluten XML-Strukturbereich beziehen soll.

Beispiel:

```
SELECT d
FROM   dokument d
WHERE  ( Vorname.content(STRING) = ‘‘Andreas’’ AND
         Nachname.content(STRING) = ‘‘Heuer’’ )
         IN d->Buch->Meta->Autor
```

Die Funktion selektiert alle Dokumente, die einen Autor enthalten, dessen Vorname "Andreas" und dessen Nachname "Heuer" lautet.

6.2.1.10 Pfadvariablen

- AS

Voraussetzung:	a : eindeutige Pfadvariable
	f : IN-Operator
Typ:	f AS *string*
Syntax:	f AS a

Durch diesen Operator werden Strukturbereiche an Pfadvariablen gebunden, falls die Prädikatfunktion des IN-Operators den Wahrheitswert *true* liefert. Pfadvariablen dürfen nur in der WHERE-Klausel nach der Auswahl eines XML-Bereiches (Abschnitt 6.2.1.2) eingeführt werden. Verwendet werden können sie dann, in der SELECT-Klausel dieses SWF-Blockes oder in der SELECT- oder WHERE-Klausel eines umgebenden SFW-Blockes.

Beispiel:

```
SELECT A
FROM   dokument d
WHERE  *.contains(‘‘Datenbank’’)
       IN d->Buch->Body->Kapitel AS A
```

Die Funktion selektiert Kapitel eines Dokumentes, die den Begriff "Datenbank" enthalten.

6.2.1.11 Strukturvergleiche

- structcompare

Voraussetzung:	$a,\ b$: absolute Pfadausdrücke
Typ:	structcompare: *xml*, *xml* $\rightarrow$ *bool*
Syntax:	$a = b$

Die Operation untersucht die Strukturbereiche a und b auf strukturelle Gleichheit, das heißt die sequentielle und hierarchische Struktur der Elemente muß identisch sein.

Beispiel:

```
SELECT d1, d2
FROM   dokument d1, dokument d2
WHERE  d1->Buch->Meta = d2->Diplomarbeit->Meta
```

Durch die Anfrage werden die Dokumente d_1 und d_2 selektiert, falls die durch d_1->*Buch*->*Meta* und d_2->*Diplomarbeit*->*Meta* beschriebenen XML-Strukturbereiche in ihrer hierarchischen und sequentiellen Struktur identisch sind.

6.2.1.12 Identität

- ident

Voraussetzung:	$a,\ b$: absolute Pfadausdrücke
Typ:	ident: *xml*, *xml* $\rightarrow$ *bool*
Syntax:	$a == b$

Die Operation untersucht die Strukturbereiche a und b auf Identität, das heißt die sequentielle und hierarchische Struktur der Elemente und die Inhalte der Elemente müssen identisch sein.

Beispiel:

```
SELECT d1
FROM   dokument d1, dokument d2
WHERE  d1->Buch->Meta == d2->Diplomarbeit->Meta
```

Durch die Anfrage wird auf Dokumente aus d_1 projiziert, falls die XML-Bereiche d_1->*Buch*->*Meta* und d_2->*Diplomarbeit*->*Meta* sowohl in der Struktur als auch im Inhalt der Elemente identisch sind.

6.2.1.13 Mengenoperationen

Die Funktionsweise der nachfolgenden Mengenoperationen soll anhand der beiden Beispielstrukturen (Struktur 1 und Struktur 2) verdeutlicht werden.

Strukur 1

```
<Buch>
   <Meta>
      <Autor>
         <Vorname>
         </Vorname>
         <Nachname>
         </Nachname>
      </Autor>
      <Verlag>
         <Verlagsname>
         </Verlagsname>
      </Verlag>
   </Meta>
</Buch>
```

Struktur 2

```
<Diplomarbeit>
   <Meta>
      <Autor>
         <Vorname>
         </Vorname>
         <Nachname>
         </Nachname>
      </Autor>
      <Schule>
         <Schulname>
         </Schulname>
      </Schule>
   </Meta>
</Diplomarbeit>
```

- union

Voraussetzung:	a, b : absolute Pfadausdrücke
Typ:	union: $xml, xml \rightarrow xml$
Syntax:	$a + b$

  Diese Operation vereinigt die Strukturbereiche a und b. Um zu gewährleisten, daß das Ergebnis ein wohlgeformtes XML-Dokument ist, werden die vereinigten Bereiche durch ein systemgeneriertes Element umschlossen.

  Beispiel:
  Buch->Meta + Diplomarbeit->Meta

  ```
  <Result>
     <Meta>
        <Autor>
           <Vorname></Vorname>
           <Nachname></Nachname>
        </Autor>
        <Verlag>
           <Verlagsname></Verlagsname>
        </Verlag>
  ```

```
  </Meta>
  <Meta>
    <Autor>
      <Vorname></Vorname>
      <Nachname></Nachname>
    </Autor>
    <Schule>
      <Schulname></Schulname>
    </Schule>
  </Meta>
</Result>
```

- difference

Voraussetzung:	$a,\ b$: absolute Pfadausdrücke
Typ:	difference: *xml*, *xml* $\rightarrow$ *xml*
Syntax:	*a - b*

Diese Operation bildet die Differenz der Strukturbereiche a und b. Um zu gewährleisten, daß das Ergebnis der Diffenrenz ein wohlgeformtes XML-Dokument ist, wird das Ergebnis durch ein systemgeneriertes Element umschlossen.

Beispiel:
Buch->Meta - Diplomarbeit->Meta

```
<Result>
  <Verlag>
    <Verlagsname></Verlagsname>
  </Verlag>
</Result>
```

- intersection

Voraussetzung:	$a,\ b$: absolute Pfadausdrücke
Typ:	intersection: *xml*, *xml* $\rightarrow$ *xml*
Syntax:	*a / b*

Diese Operation bildet die Schnittmenge der Strukturbereiche a und b. Um zu gewährleisten, daß das Ergebnis der Schnittmenge ein wohlgeformtes XML-Dokument ist, wird das Ergebnis durch ein systemgeneriertes Element umschlossen.

Beispiel:
Buch->Meta / Diplomarbeit->Meta

```
<Result>
   <Meta>
      <Autor>
         <Vorname></Vorname>
         <Nachname></Nachname>
      </Autor>
   </Meta>
   <Meta>
      <Autor>
         <Vorname></Vorname>
         <Nachname></Nachname>
      </Autor>
   </Meta>
</Result>
```

6.2.1.14 Rangordnung

- RANK BY

  Syntax: RANK BY $f_1, ..., f_n$

  Um ein globales Ranking zu erreichen wird die SFW-Anweisung um die RANK BY-Klausel erweitert. Alle, das Selektionsprädikat erfüllenden Dokumente, werden anschließend entsprechend der Ranking-Funktionen $f_1, .., f_n$ geordnet.

  Beispiel:

```
SELECT   d
FROM     dokument d
WHERE    ANY Kapitel->*.contains(“Datenbank”)
         IN d->Buch->Body
RANK BY  Titel->*.contains(“Datenbank”) IN d
```

  Durch die Anfrage werden zunächst Dokumente selektiert, die den Begriff “Datenbank” an beliebiger Position in einem Kapitel enthalten. Anschließend wird die Ordnung der Dokumente bestimmt. In der Rangordnung besser bewertet werden Dokumente, die in einem beliebigen Titel des Dokumentes den Begriff “Datenbank” beinhalten.

6.2.2 Änderungsoperationen

XML-Änderungsoperationen betreffen nicht allein das konforme Einfügen, Ändern oder Löschen eines kompletten Dokuments vom Typ *xml*. Vielmehr sollten zusätzlich Operationen zur Verfügung stehen, die es ermöglichen, Elemente, Inhalte und Attribute in Dokumente einzufügen oder innerhalb von Dokumenten zu ändern bzw. zu löschen.

In dieser Arbeit werden nur die Anweisungen für das Einfügen, Ändern und Löschen von Elementen, Attributen und Inhalten innerhalb von Dokumenten des Typs *xml* vorgestellt. Jede dieser Operationen impliziert die anschließende Prüfung des Ergebnis-Dokumentes entweder auf Konformität hinsichtlich seiner DTD oder auf Wohlgeformtheit bei Fehlen einer DTD.

6.2.2.1 Insert

- Einfügen eines XML-Bereiches

  INSERT INTO *dokument*
  AREA *expression* | [*subselect*]
  location-clause
  [WHERE *qualification*]

  Die *location-clause* bestimmt vor, nach oder unter welchem Strukturelement der angegebene Strukturbereich eingefügt werden soll. Im Unterschied zu den absoluten Pfadausdrücken der Anfrageoperationen darf dieser Pfadausdruck nur aus einem *SimpleAPE* bestehen, da der Pfadausdruck auf einen existierenden Bereich verweisen muß und nicht auf einen während der Anfrage erzeugten Bereich. Die in den folgenden Syntaxbeschreibungen der Änderungsoperationen bezeichneten absoluten Pfadausdrücke sind hier stets als *SimpleAPE* zu verstehen.

location-clause	::=	location SimpleAPE
location	::=	'UNDER'
	|	'BEFORE'
	|	'AFTER'

  In dieser Insert-Operation werden durch *expression* oder alternativ durch eine SFW-Anweisung (*subselect*) XML-Strukturbereiche angegeben, die in XML-Dokumente eingefügt werden sollen. An welcher Position das Einfügen erfolgt, wird durch die *location-clause* bestimmt. Optional kann in der WHERE-Klausel durch *qualification* eine Bedingung festgelegt werden, die erfüllt sein muß, damit der Strukturbereich in das Dokument eingefügt werden kann. Die *qualification* entspricht der *bedingung* der WHERE-Klausel der Anfrageoperationen. Eine in der *qualification* eingeführte Pfadvariable kann in der *location-clause* verwendet werden, um beispielsweise bei mehrfachem Auftreten von Strukturelementen eine eindeutige Zuordnung zu erreichen.

Beispiel:

```
INSERT INTO dokument d
AREA   “<Betreuer>
         <Vorname>Joachim</Vorname>
         <Nachname>Kröger</Nachname>
       </Betreuer>”
BEFORE A
WHERE  ( Vorname.content(STRING) = “Holger” AND
         Nachname.content(STRING) = “Meyer”)
       IN d->Diplomarbeit->Meta->Betreuer AS A
       AND NOT
       ( Vorname.content(STRING) = “Joachim” AND
         Nachname.content(STRING) = “Kröger”)
       IN d->Diplomarbeit->Meta->Betreuer
```

Durch die Beispiel-Operation wird “Joachim Kröger” als Betreuer in eine Diplomarbeit eingefügt, falls diese Diplomarbeit den Betreuer “Holger Meyer” aber noch nicht den Betreuer “Joachim Kröger” enthält. Der Strukturbereich wird, falls die Bedingung erfüllt ist, vor den Betreuer “Holger Meyer” eingefügt.

- Einfügen eines Attributs

  INSERT INTO *dokument*
  ATTRIBUTE *attributename* VALUE *expression*
  IN *absolute PathExpression*
  [WHERE *qualification*]

  Diese Operation fügt das Attribut *attributename* zusammen mit dem durch *expression* angegebenen Wert in das Wurzelelement des durch *absolute PathExpression* bestimmten Strukturbereichs ein, falls dieses Attribut noch nicht in den Attributen dieses Strukturelementes enthalten ist. Für die Operation kann optional in der WHERE-Klausel durch *qualification* eine Bedingung festgelegt werden, die erfüllt sein muß, damit das Attribut/Wert-Paar eingefügt werden kann.

  Beispiel:

```
INSERT INTO dokument d
ATTRIBUTE “Geschlecht” VALUE “w”
```

```
IN A
WHERE  ( Vorname.content(STRING) = ‘‘Beate’’ AND
         Nachname.content(STRING) = ‘‘Porst’’ )
       IN d->Diplomarbeit->Meta->Autor AS A
```

Die Beispiel-Operation fügt das Attribut *Geschlecht* sowie den Wert *w* in das Element *Autor* ein, falls es sich um den Autor “Beate Porst” handelt.

- Einfügen von Inhalten

  INSERT INTO *dokument*
  CONTENT *expression*
  IN *absolute PathExpression*
  [WHERE *qualification*]

  Diese Operation fügt durch *expression* angegebene Inhalt in das Wurzelelement des durch *absolute PathExpression* bestimmten Strukturbereichs ein. Für die Operation kann optional in der WHERE-Klausel durch *qualification* eine Bedingung festgelegt werden, die erfüllt sein muß, damit der Inhalt eingefügt werden kann. Enthält das Wurzelelement des durch *absolute PathExpression* bestimmten Strukturbereichs bereits einen Inhalt, so wird der in *expression* angegebenen Inhalt dem bestehenden Inhalt hinzugefügt.

  Beispiel:

```
INSERT INTO dokument d
CONTENT ‘‘H Information Systems’’
IN d->Diplomarbeit->Meta->Klassifikation
WHERE  ( Vorname.content(STRING) = ‘‘Beate’’ AND
         Nachname.content(STRING) = ‘‘Porst’’ )
       IN d->Diplomarbeit->Meta->Autor
```

  Die Beispiel-Operation fügt die CR-Klassifikation *H Information Systems* in den Inhalt des Elementes *Klassifikation* ein, falls es sich bei dem Dokument um die Diplomarbeit von “Beate Porst” handelt.

6.2.2.2 Update

- Ändern von Elementnamen

  UPDATE *dokument*
  ELEMENT*elementname* = *expression*
  IN *absolute PathExpression*
  [WHERE *qualification*]

Diese Operation ändert alle Elemente *elementname*, die in dem durch *absolute PathExpression* festgelegten Strukturbereich enthalten sind, auf den durch *expression* angegebenen Namen. Für die Operation kann optional in der WHERE-Klausel durch *qualification* eine Bedingung festgelegt werden, die erfüllt sein muß, damit der Elementname modifiziert werden kann.

Beispiel:

```
UPDATE dokument d
ELEMENT Obertitel = ‘‘Haupttitel’’
IN d->Buch->Meta
```

Durch die Beispiel-Operation wird der *Obertitel* von Büchern (Name des Strukturelementes) in *Haupttitel* umbenannt.

- Ändern von Attributnamen

  UPDATE *dokument*
  ATTRIBUTE *attributename* OF *relative PathExpression* = *expression*
  IN *absolute PathExpression*
  [WHERE *qualification*]

  Diese Operation ändert alle Attributnamen *attributename*, die in dem durch *relative PathExpression* und *absolute PathExpression* spezifizierten Strukturbereich vorkommen, auf den in *expression* angegebenen Namen. Für die Operation kann optional in der WHERE-Klausel durch *qualification* eine Bedingung festgelegt werden, die erfüllt sein muß, damit der Attributname modifiziert werden kann.

  Beispiel:

  ```
  UPDATE dokument d
  ATTRIBUTE ident OF * = ‘‘ID’’
  IN d->Buch
  ```

  Durch die Beispiel-Operation werden alle Attribute *ident* eines Buches in den Attributnamen “ID” geändert.

- Ändern von Attributwerten

  UPDATE *dokument*
  VALUE *attributename* OF *relative PathExpression* = *expression*
  IN *absolute PathExpression*
  [WHERE *qualification*]

Diese Operation ändert den Wert aller Attributnamen *attributename*, die in dem durch *relative PathExpression* und *absolute PathExpression* spezifizierten Strukturbereich vorkommen, auf den in *expression* angegebenen Wert. Für die Operation kann optional in der WHERE-Klausel durch *qualification* eine Bedingung festgelegt werden, die erfüllt sein muß, damit der Attributwert modifiziert werden kann.

Beispiel:

```
UPDATE dokument d
VALUE Sprache OF Buch = “en”
IN d
WHERE  Obertitel.content(STRING) = “Database”
       IN d->Buch->Meta->Titel
```

Durch die Beispiel-Operation wird der Wert des Attributs *Sprache* im Element *Buch* auf “en” geändert, falls der Obertitel des Buches “Database” lautet.

- Ändern von Inhalten

UPDATE *dokument*
CONTENT OF *elementname* = *expression*
IN *absolute PathExpression*
[WHERE *qualification*]

Diese Operation ändert den Inhalt aller Elemente *elementname*, die in dem durch *absolute PathExpression* festgelegten Strukturbereich enthalten sind, auf den in *expression* angegebenen Inhalt. Für die Operation kann optional in der WHERE-Klausel durch *qualification* eine Bedingung festgelegt werden, die erfüllt sein muß, damit der Inhalt modifiziert werden kann.

Beispiel:

```
UPDATE dokument d
CONTENT OF Obertitel = “Database”
IN A
WHERE  Obertitel.content(STRING) = “Datenbank”
       IN d->Buch->Meta->Titel AS A
```

Die Beispiel-Operation ändert den Obertitel eines Buches auf “Database”, falls der Original-Obertitel “Datenbank” lautet.

6.2.2.3 Delete

- Löschen eines XML-Bereiches

  DELETE FROM *dokument*
  AREA OF *relative PathExpression* IN *absolute PathExpression*
  [WHERE *qualification*]

  Diese Operation löscht den durch *relative PathExpression* und *absolute PathExpression* spezifizierte Strukturbereich aus dem Dokument. Für die Operation kann optional in der WHERE-Klausel durch *qualification* eine Bedingung festgelegt werden, die erfüllt sein muß, damit der Strukturbereich gelöscht werden kann.

  Beispiel:

  ```
  DELETE FROM dokument d
  AREA OF Abstract
  IN d->Diplomarbeit->Meta
  WHERE  Titel->*.contains(“XML”)
         IN d->Diplomarbeit->Meta
  ```

  Die Beispiel-Operation entfernt das Abstract einer Diplomarbeit, falls diese Diplomarbeit im Titel den Begriff “XML” beinhaltet.

- Löschen eines Attributes

  DELETE FROM *dokument*
  ATTRIBUTE *attributename* OF *relative PathExpression*
  IN *absolute PathExpression*
  [WHERE *qualification*]

  Diese Operation löscht sämtliche durch *attributename* festgelegten Attribut/Wert-Paare in dem durch *relative PathExpression* und *absolute PathExpression* spezifizierten Strukturbereich aus dem Dokument. Für die Operation kann optional in der WHERE-Klausel durch *qualification* eine Bedingung festgelegt werden, die erfüllt sein muß, damit die Attribut/Wert-Paare gelöscht werden können.

  Beispiel:

  ```
  DELETE FROM dokument d
  ATTRIBUTE ident OF *
  IN d
  ```

Die Beispiel-Operation löscht in allen Dokumenten die *ident*-Attribute.

- Löschen von Inhalten

  DELETE FROM *dokument*
  CONTENT OF *relative PathExpression* IN *absolute PathExpression*
  [WHERE *qualification*]

  Diese Operation löscht den Inhalt des durch *relative PathExpression* und *absolute PathExpression* spezifizierten Strukturbereiches aus dem Dokument. Für die Operation kann optional in der WHERE-Klausel durch *qualification* eine Bedingung festgelegt werden, die erfüllt sein muß, damit der Inhalt gelöscht werden kann.

  Beispiel:

  ```
  DELETE FROM dokument d
  CONTENT OF Klassifikation
  IN d->Diplomarbeit->Meta
  ```

  Durch die Beispiel-Operation wird der Inhalt der *Klassifikation* einer Diplomarbeit gelöscht.

6.2.3 Fazit

Für den zu Beginn dieses Abschnitts vorgestellten allgemeinen Datentyp *xml* wurde eine Syntax für Operationen vorgeschlagen, mit deren Hilfe einerseits Dokumente des Datentyps *xml* angefragt werden können, andererseits aber auch direkt XML-Bereiche oder Strukturelemente innerhalb der Dokumente eingefügt, geändert oder gelöscht werden können. Nicht betrachtet wurden hier Operationen zum Einfügen, Ändern oder Löschen von Werten des Datentyps in der Datenbank, da diese Anweisungen sich nicht von den SQL-Anweisungen unterscheiden. Es müssen nur Integritätsbedingungen berücksichtigt werden, die u.a. die Prüfung des Dokumentes auf Wohlgeformtheit oder Gültigkeit beinhalten. In dem Vorschlag des allgemeinen Datentyps wurde desweiteren nicht darauf eingegangen, wie beispielsweise die Indizierung der XML-Dokumente zu erfolgen hat und wie ein Volltext-Retrieval auf temporär konstruierten Dokumenten ausgeführt werden kann. Auch die Behandlung von Ausnahmen, die beispielsweise durch Fehler bei der Typkonvertierung ausgelöst werden müssen, wurde in dieser Arbeit nicht diskutiert. Trotz des großen Funktionsumfangs ist die Vollständigkeit nicht gesichert. Die Funktionen des Datentyps sind allerdings modular aufgebaut, so daß die Skalierung der Funktionalität jederzeit möglich ist.

6.3 Zusammenfassung

In Abschnitt 6.1 wurden existierende XML-Anfragesprachen vorgestellt, um herauszustellen, welche Funktionalität durch die einzelnen Sprachen unterstützt wird. Insbesondere die Bereiche Retrieval, Ranking und objekterhaltende Änderungen der Dokumente finden derzeit keine Berücksichtigung in Form von Operationen dieser Sprachen. Das Ziel bestand darin, daß der zu konzipierende Datentyp *xml* alle in der Tabelle 6.1 zusammengefaßten Funktionen integriert. Diese Zielsetzung wurde durch den in Abschnitt 6.2 unterbreiteten Vorschlag des allgemeinen Datentyps erreicht. Zusätzlich können aufgrund der Erweiterung der SQL-Syntax Anfragen an XML-Dokumente mit "gewöhnlichen" Attributanfragen kombiniert werden. Insofern kann der Vorschlag als "ideal" betrachtet werden.

Kapitel 7

Schlußbetrachtung

Aktuelle Entwicklungen zeigen, daß der Informationsaustausch durch semistrukturierte Dokumente, deren implizite Struktur aus XML-Strukturkonstrukten besteht, eine deutliche Zunahme erfährt. Einher mit dieser Tendenz geht die Herausforderung, diese Dokumente nicht nur durch Datenbanken zu verwalten, sondern mittels Retrieval-Systemen aus diesen Dokumenten gezielt Informationen zu gewinnen, wobei die implizite Struktur berücksichtigt wird.

Das Ziel dieser Arbeit war es nun, sich dieser Herausforderung zu stellen. Als Ansatzpunkt wurden die objektrelationalen Datenbanksysteme DB2 und Informix verwendet, die aufgrund ihrer Erweiterbarkeit die Definition nutzerdefinierter Datentypen und Datenbankroutinen erlauben und somit die Verlagerung von Anwendungslogik auf den Datenbankserver ermöglichen. Ein weiterer Aspekt, der für die Nutzung ORDBS spricht, sind Retrieval-Erweiterungen (DB2 TextExtender und Excalibur DataBlade), die bereits einen integralen Bestandteil dieser Systeme bilden.

7.1 Zusammenfassung

Die Funktionalität der beiden Volltext-Retrieval-Module wurde zunächst unabhängig von den Anforderungen semistrukturierter Dokumente evaluiert. Das Resultat der Untersuchung und Gegenüberstellung sprach dabei klar für den TextExtender von DB2 als Volltext-Retrieval-System. Das Datenmodell von DB2 ist jedoch gegenüber dem Informix-Datenmodell als deutlich schwächer einzuschätzen. Die generelle Verwendbarkeit dieser Retrieval-Systeme hinsichtlich der Anforderungskriterien semistrukturierter Dokumente wurde analysiert, wobei u.a. die Möglichkeit diskutiert wurde, ob durch die Abbildung der impliziten Struktur auf die Datenbankstruktur die vorhandenen Retrieval-Systeme direkt genutzt werden können. Argumente wie der Informationsverlust durch fehlende

Typisierung, die problematische Rekonstruktion der Originaldokumente und die schwierige Realisierung von Anfragen, die das Schema vernachlässigen, sprachen aber gegen diese Überlegungen. Aufgrund der Beschränkung des Anwendungsbereiches auf die Domäne "Digitale Bibliothek" wurde statt dessen ein hybrider Ansatz gewählt, der nur Metainformationen der Dokumente als Attribute des Datenbankschemas repräsentiert. Anfragen nach Informationen, die nicht in den Meta-Attributen enthalten sind, werden hingegen direkt aus den als Volltext gespeicherten Dokumenten gewonnen. Da beim Information-Retrieval auf semistrukturierten Dokumenten der Differenzierung zwischen Struktur und Inhalt eine große Bedeutung beigemessen wird, die vorhandenen Retrieval-Systeme diese Unterscheidung aber nicht unterstützen, wurde eine Erweiterung des Suchausdrucks des TextExtenders konzipiert, der diese Anforderung integriert. Durch die prototypische Implementierung der konzipierten Konstrukte des TextExtenders wurden die Möglichkeiten und Grenzen der Realisierung eines XML-Retrievals auf Basis eines erweiterten Volltext-Retrieval-Moduls im ORDBS DB2 verdeutlicht. Gezeigt werden konnte, daß eine auf dem TextExtender basierende XML-Erweiterung realisierbar ist. Das angesichts des fehlenden Strukturindex notwendige sequentielle Lesen sowie die starke Einschränkung der linguistischen Funktionalität beeinträchtigen indessen die Weiterentwicklung und Verwendung unter realen Bedingungen. Beeinträchtigt wird der Ausbau des XML-Retrieval-Moduls auch durch konzeptionell bedingte Festlegungen, wie der Orientierung der XML-Syntax an der Syntax des Volltext-Retrieval-Systems. Da auch andere existierende XML-Anfragesprachen nur eine Minimal- oder Teilfunktionalität unterstützen, wurde als Konsequenz ein spezieller XML-Datentyp vorgeschlagen. Dieser umfaßt nicht nur die in den existierenden Anfragesprachen vorhandenen Operationen sondern zusätzlich notwendige Operationen für die objekterhaltende Modifikation, Retrieval und Ranking von XML-Dokumenten.

7.2 Ausblick

In Kapitel 1 wurde der Bezug zwischen dieser Arbeit und laufenden Projekten am Lehrstuhl für Datenbank- und Informationssysteme gezogen. Insbesondere die Konzeption und Implementierung des XML-Retrieval-Moduls auf der Grundlage der TextExtender Volltext-Erweiterung kann als Ausgangspunkt der Entwicklung des Lokalen Dokumenten Servers fungieren. Notwendige Arbeiten zur Realisierung bestehen dann u.a. aus:

- der Umsetzung der vollen Funktionalität der Konzeption inklusive der Integritätsbedingungen für Änderungsoperationen,
- der Erweiterung des Datenbankentwurfs um weitere Objekttypen einer "Digitalen Bibliothek",

- der Vervollständigung der DTD-Beschreibungen,
- der Speicherung der DTD-Beschreibungen in der Datenbank und
- der Konzeption und Implementierung weiterer Funktionalitäten, wie beispielsweise der Extraktion von Werten aus den Dokumenten, um Aggregationen auf Elementinhalten zu ermöglichen.

Die Nachteile dieses auf dem TextExtender basierenden Ansatzes wurden bereits mehrfach diskutiert. Eine maßgebliche Verbesserung des Zeitverhaltens kann durch die Implementierung von Zugriffspfaden erzielt werden, die eine Trennung von Struktur und Inhalt berücksichtigen. Da sich inzwischen auch Datenbankhersteller der Thematik XML angenommen haben, ist in Kürze mit neuen Releases zu rechnen, die Speicherung, Retrieval und Anfragen von XML in einem Datenbanksystem integrieren.[1] Statt der Adaptierung einer existierenden Datenbank an XML wurde durch die Software AG der Weg über einen eigenständigen XML-Information-Server (Tamino[2]) gegangen. Taminos Kerntechnologie beruht auf der X-Machine, in der XML-Informationen ohne Konvertierung in andere Strukturen gespeichert werden können. Über X-Nodes kann von Tamino aus auf existierende Datenbanken und Anwendungen zugegriffen und die in den Datenbanken enthaltenen Informationen auf XML-Strukturen abgebildet werden. Es ist daher sinnvoll, die Erweiterung des TextExtenders nur als Zwischenlösung zu betrachten, die bei Verfügbarkeit durch eines der Systeme (Tamino oder ein Datenbanksystems mit XML-Unterstützung) abgelöst wird.

[1] Oracle kündigte an, in der Datenbank Oracle8i zum Ende des Jahres 1999 XML zu unterstützen. IBM möchte bereits bis Mitte dieses Jahres eine XML-Erweiterung für DB2 fertigstellen.

[2] http://www.softwareag.com/tamino/

Literaturverzeichnis

[Abi97] Abiteboul, S.: Querying semi-structured data. In: *Proceedings of the International Conference on Database Theory*, 1997.

[Alm98] Alm, L.: Untersuchung zur Kopplung von Datenbanken und IR-Techniken. Diplomarbeit, Universität Rostock, Fachbereich Informatik, 1998.

[AQMW97] Abiteboul, S.; Quas, D.; McHugh, J.; Widom, J.: The lorel query language for semistructured data. In: *International Journal on Digital Libraries*, S. 68–88, April 1997.

[BDFS97] Buneman, P.; Davidson, S.; Fernandez, M.; Suziu, D.: Adding structure to unstructured data. In: *Proceedings of the International Conference on Database Theory*, 1997.

[BDH96] Buneman, P.; Davidson, S.; Hillebrand, G.: A query language and optimization techniques for unstructured data. In: *Proc. of the ACM SIGMOD International Conference on Management of Data*, 1996.

[BDS95] Buneman, P.; Davidson, S.; Suciu, D.: Programming constructs for unstructured data. In: *Proc. Workshop on Database Programming Languages*, 1995.

[BM98] Behme, H.; Mintert, S.: *XML in der Praxis.* Addison-Wesley, 1. Auflage, 1998.

[BPSM98] Bray, T.; Paoli, J.; Sperberg-McQueen, C.: Extensible Markup Language (XML) 1.0, Februar 1998. `http://www.w3.org/TR/1998/REC-xml-19980210`.

[Bra98] Bradley, N.: *The XML companion.* Addison Wesley, Harlow, England, 1. Auflage, 1998.

[Cha98] Chamberlin, D.: *A complete guide to DB2 Universal Database.* Morgan Kaufmann Publishers, San Francisco, 1. Auflage, 1998.

[Clu97] Cluet, S.: Modeling and querying semi-structured data. *Lecture Notes in Computer Science*, Band 1299, S. 192–213, 1997.

[Dao98] Dao, T.: An indexing model for structured documents to support queries on content, structure and attributes. In: *Proceedings of the IEEE ADL*, S. 88–97, 1998.

[DeR98] DeRose, S.: Xquery: A unified syntax for linking and querying general XML documtents, Oktober 1998. `http://www.w3.org/TandS/QL/QL98/pp/xquery.html`.

[DFF+98] Deutsch, A.; Fernandez, M.; Florescu, D.; Levy, A.; Suciu, D.: XML-QL: A query language for XML, August 1998. `http://www.w3.org/TR/1998/NOTE-xml-ql-19980819/`.

[FLM98] Florescu, D.; Levy, A.; Mendelzon, A.: Database techniques for the world wide web: A survey. *SIGMOD Record*, Band 27, S. 59–74, September 1998.

[Fuh97] Fuhr, N.: Information Retrieval, Skriptum zur Vorlesung, 1997. `http://ls6-www.informatik.uni-dortmund.de/ir/doc/courses/ir/irskall.ps.gz`.

[Gol90] Goldfarb, C. F.: *The SGML Handbook*. Clarendon Press, Oxford, 1990.

[Heu98] Heuer, A.: Global-Info Rahmenantrag auf eine Sonderfördermaßnahme im Schwerpunkt 4, 1998. `http://wwwdb.informatik.uni-rostock.de/global-info/bluerose/index.html`.

[HS95] Heuer, A.; Saake, G.: *Datenbanken*. Thomson Publishing, 1. Auflage, 1995.

[IBM97a] IBM Corporation: *DB2 Universal Database Embedded SQL-Programming Guide*, September 1997.

[IBM97b] IBM Corporation: *DB2 Universal Database TextExtender Administration and Programming*, September 1997.

[Inf97] Informix Software Inc.: *Excalibur Text Search DataBlade Module*, Juli 1997.

[Kop96] Kopka, H.: *LATEX Einführung*. Addison-Wesley, Bonn, 2. Auflage, 1996.

[Pet98] Petkovic, D.: *INFORMIX Universal Server*. Addison-Wesley, Bonn, 1998.

[Rah98] Rahm, E.: Evaluation of object-relational database systems for full-text retrieval, 1998. Report Nr. 6.

[RLS98] Robie, J.; Lapp, J.; Schach, D.: XML query language (XQL), December 1998. `http://www.w3.org/TandS/QL/QL98/pp/query-transform.html`.

[Sal71] Salton, G.: *The SMART Retrieval System - Experiments in automatic document processing.* Prentice Hall, Englewood Cliff, New Jersey, 1971.

[Sch98] Schmiede, R.: Global-Info Ein neuer Fokus für Digital Library Entwicklungen, 1998. `http://www.global-info.org/doc/schmiede-9801.html`.

[Se98] Staab, S.; et.al.: A system for facilitating and enhancing web search, 1998. `http://wwwdb.informatik.uni-rostock.de/~duest/ueberblick.dvi.ps`.

[Sto96] Stonebraker, M.: *Object-Relational DBMSs The next great wave.* Morgan Kaufmann Publishers, San Francisco, 1996.

Abbildungsverzeichnis

Tabellenverzeichnis

Anhang A

Beispiel-DTD

Durch die nachfolgenden Dokumentenbeschreibungen werden die Namen der Elemente, Attribute und deren logischen Schachtelung für die Objekttypen *Buch* und *Diplomarbeit* der “Digitalen Bibliothek” definiert. Es handelt sich dabei nur um Beispiele, die speziell im *Body*-Bereich keine Vollständigkeit aufweisen. Die Definition der Beschreibung verfolgt daß Ziel, die in der Arbeit vorgestellten Beispiele zu begleiten, die Möglichkeiten für die Definition einer DTD vorzustellen und die zum Test der Implementierung verwendeten Dokumente als Instanzen dieser DTDs abzuleiten.

A.1 Buch DTD

```
<!ELEMENT DOCTYPE (Buch)>

<!ENTITY % Block “Liste | Tabelle | Graphic | Para“>

<!ELEMENT Buch (Meta,Body,Ende?)>
<!ATTLIST Buch Sprache xml:lang NMTOKEN “de“>

<!ELEMENT Meta (Titel, (Autor+ | Editor+), Verlag, Jahr, ISBN,
                Edition?, Volume?, Note?, Copy?)  >

<!ELEMENT Body ( (Vorwort, (Teil | Kapitel+) ) |
                 (Teil | Kapitel+) )>

<!ELEMENT Ende (Anhang*, Literatur?)>

<!-- Elemente des Tags “Meta“-->
```

```
<!ELEMENT Autor (Vorname, Middlename?, Nachname)>
<!ELEMENT Editor (Vorname, Middlename?, Nachname)>
<!ELEMENT Edition     (#PCDATA)>
<!ELEMENT Verlag      (Verlagsname, Adresse?)>
<!ELEMENT Verlagsname (#PCDATA)>
<!ELEMENT Adresse     (Strasse, Plz, Ort)>
<!ELEMENT Strasse     (#PCDATA)>
<!ELEMENT Plz         (#PCDATA)>
<!ELEMENT Ort         (#PCDATA)>
<!ELEMENT ISBN        (#PCDATA)>
<!ELEMENT Jahr        (#PCDATA)>
<!ELEMENT Volume      (#PCDATA)>
<!ELEMENT Note        (#PCDATA)>
<!ELEMENT Copy        (#PCDATA)>
<!ELEMENT Vorname     (#PCDATA)>
<!ELEMENT Middlename  (#PCDATA)>
<!ELEMENT Nachname    (#PCDATA)>

<!-- Elemente des Tags “Body“ -->

<!ELEMENT Vorwort (%Block;)+>
<!ELEMENT Teil ((Titel, (%Block;)*, Kapitel+)+)>
<!ATTLIST Teil ident ID #REQUIRED>

<!ELEMENT Kapitel (Titel, (%Block;)*, Abschnitt*)>
<!ATTLIST Kapitel ident ID #REQUIRED>

<!ELEMENT Abschnitt (Titel, (%Block;)*, Unterabschnitt*)>
<!ATTLIST Abschnitt ident ID #REQUIRED>

<!ELEMENT Unterabschnitt (Titel, (%Block;), Unterabschnitt*)>
<!ATTLIST Unterabschnitt ident ID #REQUIRED>

<!-- Elemente des Tags “Ende“ -->

<!ELEMENT Anhang (Titel, (%Block;)* Abschnitt*)>
<!ATTLIST Anhang ident ID #REQUIRED>

<!ELEMENT Literatur (Eintrag+)>
<!ELEMENT Eintrag (Autor+, Titel )>
<!ATTLIST Eintrag ref ID #REQUIRED>

<!-- Elemente des Entity “Block“ -->
```

```
<!ELEMENT Para (#PCDATA | Citation | Reference)*>
<!ELEMENT Graphic EMPTY>
<!ATTLIST Graphic id ID #IMPLIED
          ident ENTITY #REQUIRED>

<!ELEMENT Liste (Item+)>
<!ELEMENT Item (#PCDATA | Citation | Reference)*>
<!ATTLIST Item listtype (number | random) “random”>

<!ELEMENT Tabelle (Ueberschrift?, Zeile*, Unterschrift?)>
<!ELEMENT Ueberschrift (#PCDATA)>
<!ELEMENT Zeile (Spalte+)>
<!ATTLIST Zeile Nummer ID #REQUIRED>

<!ELEMENT Spalte (#PCDATA | Citation | Reference)*>
<!ELEMENT Unterschrift (#PCDATA)>

<!-- Elemente verschiedener Tags -->

<!ELEMENT Titel (Obertitel, Untertitel?)>
<!ELEMENT Obertitel (#PCDATA)>
<!ELEMENT Untertitel (#PCDATA)>
<!ELEMENT Citation (#PCDATA)>
<!ATTLIST Citation cite IDREF #REQUIRED>
<!ELEMENT Reference (#PCDATA)>
<!ATTLIST Reference ref IDREF #REQUIRED>
```

A.2 Diplomarbeit DTD

```
<!ELEMENT DOCTYPE (Diplomarbeit)>

<!ENTITY % Block “Liste | Tabelle | Graphic | Para“>

<!ELEMENT Diplomarbeit (Meta,Body,Ende?)>
<!ATTLIST Diplomarbeit Sprache xml:lang NMTOKEN “de“>

<!ELEMENT Meta (Titel, Autor, Jahr, Schule, Fachbereich, Betreuer+,
                Bearbeitungszeit?, Abstract?, Keywords?,
                Klassifikation?)>
```

```
<!ELEMENT Body ( (Vorwort, (Teil | Kapitel+) ) |
                 (Teil | Kapitel+) )>

<!ELEMENT Ende (Anhang*, Literatur?)>

<!-- Elemente des Tags ``Meta`` -->

<!ELEMENT Autor (Vorname, Middlename?, Nachname)>
<!ELEMENT Jahr             (#PCDATA)>
<!ELEMENT Schule           (Schulname, Adresse?)>
<!ELEMENT Schulname        (#PCDATA)>
<!ELEMENT Adresse          (Strasse, Plz, Ort)>
<!ELEMENT Strasse          (#PCDATA)>
<!ELEMENT Plz              (#PCDATA)>
<!ELEMENT Ort              (#PCDATA)>
<!ElEMENT Fachbereich      (#PCDATA)>
<!ELEMENT Betreuer (PersTitel?, Vorname, Middlename?, Nachname)>
<!Element PersTitel        (#PCDATA)>
<!ELEMENT Bearbeitungszeit (Vom, Bis)>
<!ELEMENT Vom              (#PCDATA)>
<!ELEMENT Bis              (#PCDATA)>
<!ELEMENT Abstract         (#PCDATA)>
<!ELEMENT Keywords         (Begriff+)>
<!ELEMENT Begriff          (#PCDATA)>
<!ELEMENT Klassifikation   (#PCDATA)>
<!ELEMENT Vorname          (#PCDATA)>
<!ELEMENT Middlename       (#PCDATA)>
<!ELEMENT Nachname         (#PCDATA)>

<!-- Elemente des Tags ``Body`` -->

<!ELEMENT Vorwort (%Block;)+>
<!ELEMENT Teil ((Titel, (%Block;)*, Kapitel+)+)>
<!ATTLIST Teil ident ID #REQUIRED>

<!ELEMENT Kapitel (Titel, (%Block;)*, Abschnitt*)>
<!ATTLIST Kapitel ident ID #REQUIRED>

<!ELEMENT Abschnitt (Titel, (%Block;)*, Unterabschnitt*)>
<!ATTLIST Abschnitt ident ID #REQUIRED>

<!ELEMENT Unterabschnitt (Titel, (%Block;), Unterabschnitt*)>
<!ATTLIST Unterabschnitt ident ID #REQUIRED>
```

```
<!-- Elemente des Tags ‘‘Ende‘‘ -->

<!ELEMENT Anhang (Titel, (%Block;)* Abschnitt*)>
<!ATTLIST Anhang ident ID #REQUIRED>

<!ELEMENT Literatur (Eintrag+)>
<!ELEMENT Eintrag (Autor+, Titel )>
<!ATTLIST Eintrag ref ID #REQUIRED>

<!-- Elemente des Entity ‘‘Block‘‘ -->

<!ELEMENT Para (#PCDATA | Citation | Reference)*>
<!ELEMENT Graphic EMPTY>
<!ATTLIST Graphic id ID #IMPLIED
          ident ENTITY #REQUIRED>

<!ELEMENT Liste (Item+)>
<!ELEMENT Item (#PCDATA | Citation | Reference)*>
<!ATTLIST Item listtype (number | random) ‘‘random’’>

<!ELEMENT Tabelle (Ueberschrift?, Zeile*, Unterschrift?)>
<!ELEMENT Ueberschrift (#PCDATA)>
<!ELEMENT Zeile (Spalte+)>
<!ATTLIST Zeile Nummer ID #REQUIRED>

<!ELEMENT Spalte (#PCDATA | Citation | Reference)*>
<!ELEMENT Unterschrift (#PCDATA)>

<!-- Elemente verschiedener Tags -->

<!ELEMENT Titel (Obertitel, Untertitel?)>
<!ELEMENT Obertitel (#PCDATA)>
<!ELEMENT Untertitel (#PCDATA)>
<!ELEMENT Citation (#PCDATA)>
<!ATTLIST Citation cite IDREF #REQUIRED>
<!ELEMENT Reference (#PCDATA)>
<!ATTLIST Reference ref IDREF #REQUIRED>
```

Anhang B

Syntax des TextExtender Suchargumentes

SearchArgument	::=	(ThesaurusDef)? (Result)? BooleanArgument ('&' Freetext)
	|	(ThesaurusDef)? (Result)? (BooleanArgument '&')? Freetext
BooleanArgument	::=	SearchFactor
	|	SearchFactor OP BooleanArgument
	|	'(' BooleanArgument1 ')'
BooleanArgument1	::=	BooleanArgument1 OP SearchFactor
	|	SearchFactor OP SearchFactor
Freetext	::=	'IS ABOUT' Condition Phrase
SearchFactor	::=	('NOT')? SearchPrimary
	|	Proximity
Proximity	::=	SearchPrimary 'IN SAME SENTENCE AS' SearchPrimary ('AND' SearchPrimary)*
	|	SearchPrimary 'IN SAME PARAGRAPH AS' SearchPrimary ('AND' SearchPrimary)*
SearchPrimary	::=	SearchAtom
	|	'(' SearchPrimary1 ')'
SearchPrimary1	::=	SearchAtom
	|	',' SearchPrimary1
SearchAtom	::=	SearchAtom1 Language? Phrase Escape?

SearchAtom1	::=	Precise
	|	Stemmed
	|	Fuzzy
	|	Synonym
	|	Sound
	|	Bound
	|	Thesaurus
Precise	::=	'PRECISE FORM OF'
Stemmed	::=	'STEMMED FORM OF'
Fuzzy	::=	'FUZZY FORM OF'
Synonym	::=	'SYNONYM FORM OF'
Sound	::=	'SOUNDS LIKE'
Bound	::=	'BOUND'
Result	::=	'RESULT LIMIT' number
ThesaurusDef	::=	'THESAURUS' thes-name ('COUNT' depth)?
Thesaurus	::=	('EXPAND' Thesaurusrelation)? 'TERM OF'
Phrase	::=	" Phrase oder Satz "
Language	::=	'GERMAN'
	|	'US_ENGLISH'
	|	'UK_ENGLISH'
	|	weitere Sprachen (siehe [IBM97b])
Escape	::=	'ESCAPE' " Escape-Zeichen "
OP	::=	'&'
	|	'|'
Condition	::=	Synonym
	|	Thesaurus

Anhang C

Konzeptionelle Syntax der XML-Erweiterung von DB2

Die nachfolgenden EBNF-Regeln beschreiben die konzeptionelle Syntax, die den TextExtender um Konstrukte für das Retrieval auf Struktur und Inhalt erweitert. Beim Ausbau der Syntax wurde gezielt versucht, die Orientierung an der Syntax des TextExtenders beizubehalten (siehe Anhang B). Die prototypische Implementierung setzt zur Zeit nicht die Regeln *Counter, Neighbour, Type* und *Result* um, so daß keine Typkonvertierungen, Inklusionsoperationen oder Abstandsanfragen auf Elementen ausgeführt werden können.

```
SearchArgument    ::= (Result)? BooleanArgument

BooleanArgument   ::= SearchFactor
                    | SearchFactor OP BooleanArgument
                    | '(' BooleanArgument1 ')'

BooleanArgument1  ::= BooleanArgument1 OP SearchFactor
                    | SearchFactor OP SearchFactor

SearchFactor      ::= ('NOT')? SearchPrimary
                    | 'TERMS' BaseExpression 'IN SAME ELEMENT AS'
                      SearchFactor1

SearchPrimary     ::= SearchAtom
                    | '(' SearchPrimary1 ')'

SearchPrimary1    ::= SearchAtom
                    | ',' SearchPrimary1
```

```
SearchFactor1      ::= SearchPrimary2
                     | SearchPrimary2 'AND' SearchFactor1

SearchPrimary2     ::= SimpleSearch
                     | '(' SearchPrimary3 ')'

SearchPrimary3     ::= SimpleSearch
                     | ',' SearchPrimary3

BaseExpression     ::= BaseSearch
                     | '(' BaseExpression1 ')'

BaseExpression1    ::= BaseSearch
                     | ',' BaseExpression1

SearchAtom         ::= SimpleSearch
                     | ContentSearch
                     | ElementSearch
                     | AttributeSearch

BaseSearch         ::= SimpleSearch
                     | ContentSearch

ContentSearch      ::= 'CONTENT' Compare? SimpleSearch Type?
                       PathExpression? Counter?
                     | 'CONTENT' Compare? SimpleSearch Type?
                       PathExpression? Neighbour?

ElementSearch      ::= 'ELEMENT' SimpleSearch PathExpression? Counter?
                     | 'ELEMENT' SimpleSearch PathExpression? Neighbour?

AttributeSearch    ::= 'ATTRIBUTE' AttributeValue1 Type PathExpression?

PathExpression     ::= 'IN' PathExpression1

PathExpression1    ::= Path AttributeDef? '.' PathExpression1
                     | IterateOp '.' PathExpression1

AttributeDef       ::= 'WITH' AttributeValue

AttributeValue     ::= AttributeValue1
                     | AttributeValue1 'AND' AttributeValue
```

```
AttributeValue1   ::= Attribute (Compare Value)?

Path              ::= SimpleSearch

Attribute         ::= SimpleSearch

Value             ::= SimpleSearch

SimpleSearch      ::= SimpleSearchAtom

SimpleSearchAtom  ::= (’PRECISE FORM OF’)? Word

Counter           ::= ’COUNT’ CounterNumber

Neighbour         ::= Direction CounterNumber

Direction         ::= ’LEFT’
                    | ’RIGHT’
                    | ’DOWN’
                    | ’TOP’

CounterNumber     ::= ’(’ Number ’)’

OP                ::= ’&’
                    | ’|’

Compare           ::= ’>’
                    | ’<’
                    | ’=’

IterateOp         ::= ’*’
                    | ’+’
                    | ’?’

Type              ::= ’TYPE’ DataType

DataType          ::= ’Int’
                    | ’Real’
                    | ’Char’
                    | ’Date’

Result            ::= ’RESULT LIMIT’ Number
```

Word	::=	"Phrase oder Satz"
Number	::=	[1-9][0-9]*